PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 :
H04L 12/24, H04M 3/24

A1

(11) International Publication Number: WO 94/06232

(43) International Publication Date: 17 March 1994 (17.03.94)

(21) International Application Number: PCT/SE93/00687

(22) International Filing Date: 18 August 1993 (18.08.93)

(30) Priority data:
9202488-4 28 August 1992 (28.08.92) SE
9300363-0 5 February 1993 (05.02.93) SE

(71) Applicant: TELEFONAKTIEBOLAGET LM ERICSSON [SE/SE]; S-126 25 Stockholm (SE).

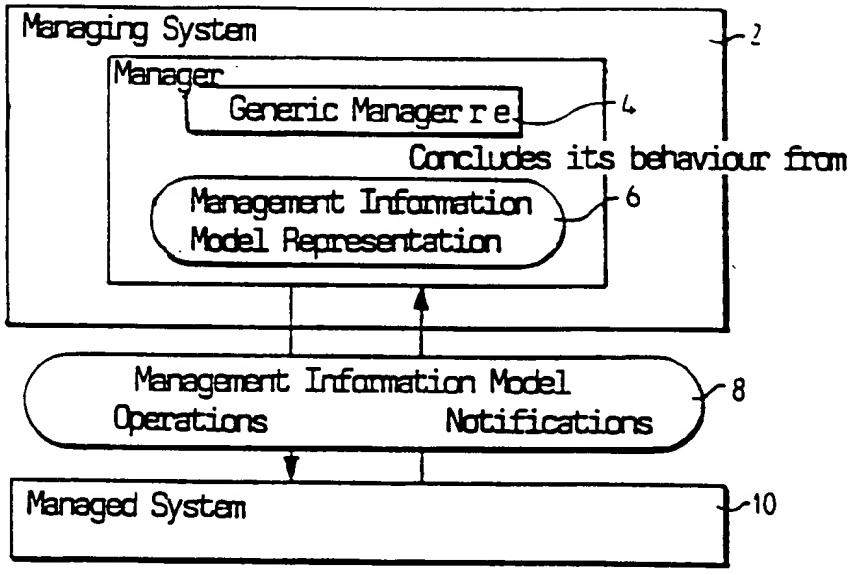
(72) Inventors: CAREBRAND, Per-Arne ; Tantogatan 43, S-118 42 Stockholm (SE). SVEDBERG, Johan ; Värtavägen 6, nb., S-115 24 Stockholm (SE). FANTENBERG, Johan ; Alströmergatan 32:6, S-112 47 Stockholm (SE). TALLDAL, Björn ; Stopvägen 38, S-161 46 Bromma (SE). PÅLSSON, Martin ; Månstorpsvägen 22, S-146 45 Tullinge (SE). GILANDER, Anders ; Sleipnergatan 46, S-195 54 Märsta (SE). SELLSTEDT, Patrik ; Lillhagsvägen 22, S-124 71 Bandhagen (SE). STRÖMBERG, Stefan ; Röntgenvägen 1, S-141 52 Huddinge (SE).

(74) Agents: DELHAGE, Einar et al.; Bergenstråhle & Lindvall AB, Box 17704, S-118 93 Stockholm (SE).

(81) Designated States: AU, BR, CA, FI, JP, KR, NO, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: MANAGEMENT IN TELECOM AND OPEN SYSTEMS



(57) Abstract

The invention relates to a management network with at least one managing system (2) and at least one managed system (10), for telecom or open systems. Said managed system includes physical and/or logical resources, which by the managing system are considered and managed as managed objects in the form of data images of the resources. The managing system utilizes for its operations directed towards the managed system a management information model (8) of the managed system, which includes a description of all managed objects, adapted to the mode of operation of the managing system. The management network includes, besides the managed system (10), a generic manager (4) and a representation (6) of the management information model, where the performance of the generic manager during operation is determined by this model representation.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FR	France	MR	Mauritania
AU	Australia	GA	Gabon	MW	Malawi
BB	Barbados	GB	United Kingdom	NE	Niger
BE	Belgium	GN	Guinea	NL	Netherlands
BF	Burkina Faso	GR	Greece	NO	Norway
BG	Bulgaria	HU	Hungary	NZ	New Zealand
BJ	Benin	IE	Ireland	PL	Poland
BR	Brazil	IT	Italy	PT	Portugal
BY	Belarus	JP	Japan	RO	Romania
CA	Canada	KP	Democratic People's Republic of Korea	RU	Russian Federation
CF	Central African Republic			SD	Sudan
CG	Congo	KR	Republic of Korea	SE	Sweden
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovak Republic
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LU	Luxembourg	TD	Chad
CS	Czechoslovakia	LV	Latvia	TG	Togo
CZ	Czech Republic	MC	Monaco	UA	Ukraine
DE	Germany	MG	Madagascar	US	United States of America
DK	Denmark	ML	Mali	UZ	Uzbekistan
ES	Spain	MN	Mongolia	VN	Viet Nam
FI	Finland				

Management in telecom and open systems

Technical field

The present invention relates to a management network with at least one managing system and at least one managed system for telecom or open systems, in which said managed system includes physical and/or logical resources, which by the managing system are considered and managed as managed objects in the form of data images of the resources, and in which the managing system for its operations directed towards the managed system utilizes an information model of the managed system, which includes a description of all managed objects adapted to the mode of operation of the managing system.

The invention also relates to a method for implementing a managed object in a management network with at least one managing system and at least one managed system, for telecom or open systems, wherein by subsystem is meant each part of a managed system, which includes one or more managed objects.

With a "management network with at least one managing system and at least one managed system" is meant that the management network can include at least one managing system which can manage one or more managed systems, which likewise can form part of the management network.

With an open system is meant a system of the kind, which is defined in Reference Model of Open Systems Interconnection (OSI) for CCITT Applications, Rec. X.200.

To perform management activities in a management domain there must be at least one manager which is responsible for the management of the resources. A resource is something which includes concepts and ability of the domain. An example of a domain is a telecom network, where the resources are switches, trunks etc. and management units are e.g. operator tools and managing systems for the network.

For operation of telephone networks each individual company has used a number of different systems for operation

and maintenance. CCITT has developed a standard model for operation and maintenance in telephone networks, called TMN (Telecommunication Management Networks). The basic principle of TMN is to indicate an organised network structure, which admits connection of various managing systems to telecom equipment. This is achieved by use of standardised protocols and interfaces. The telephone companies and other operators will require that future telecom networks are adapted to TMN.

CCITT has a recommendation for this under developement, the M.3000-serie.

TMN considers all network nodes as network elements (NE). These network elements are made as telephone switches and transmission or transport network products.

The functional structure of TMN comprises

- management functions (OSF, Operations Support Functions), which manage application programmes available for users, such as management functions for "Business Management" and service and network administration;
- data communication functions (DCF; Data Communications Functions), which manage data communication between the managing systems OSS and the managed systems NE;
- mediation functions (MF, Mediation Functions), which convert information (between e.g. managed objects), manage data, concentrate, reduce and edit, make decisions relating to e.g. threshold limits and store data, which identify equipment and networks;
- network element functions (NEF, Network Element Functions), which manage telecom processes as switching functions and transmission, and take part in management processes for telecommunication, as fault localisation and protection connections;
- functions for interface adaption (QAF, Q-Adapter Functions), which perform conversion of interfaces from non standard to standard;
- work station functions (WSF, Work Station Functions), which constitute the user terminals of TMN, show information and assist management technicians to manage the network.

TMN includes also an interface, called Q3. Q3 is besides being a communication protocol also an information model,

which comprises data schemes, operations and notifications. The exact details of the Q3 interface and its protocols appear from the CCITT recommendations Q961 and Q962.

TMN considers all physical and logical objects as managed objects, which in TMN are referred to as MO (Managed Objects), a denomination which will be used alternately also here henceforth. Managed objects are data images of such physical or logical resources as wires, circuits, signal terminals, transmission routes, events logs, alarm reports etc..

A specific relationship occurs between a resource and a managed object. A resource can be related to one or more MO or none at all. On the other hand an MO can be related to one or more resources or none at all. This relationship is important if an MO is affected by some form of operation or maintenance activity. A MO must not be removed before the functions to which it is subordinated have themselves been removed. This information model is based upon object orientation and the relation concept.

The managing system (OSS- Operation Support system) treats network elements and subordinated managing systems as a collection of managed objects in an imagined data base MIB (Management Information Base). These managed objects are constituted by instances of an MO class, such as a number of signal terminals of the same type. Each terminal will thus constitute an instance of the class signal terminals.

In TMN exists also the concept MIM (Management Information Model), which refers collectively to all information related to managed objects. MIM is a model of all attributes, relations, operations and notifications referable to managed objects. To be able to search for MO-instances a management information tree MIT (Management Information Tree) is used. This tree structure starts in the network and indicates network elements and subscribers or equipment.

For operation and maintenance each seperate unit or resource, of which information is required, or the operation of which is influenced from the outside of the system, is represented by a managed object. Each information exchange

referable to the management of operations or units, which must be influenced or which must report something (such as set up of data, assignement of name, or management of alarms) is done in the form of operations on, or notes from managed objects.

A managed object includes attributes and can also have a relation to other managed objects. A number of different operations can be directed towards a managed object and events can be generated by such objetcs.

Below an account of deficiencies of TMN will be given.

Network elements and subordinated managing systems are managed by a managing system by means of operations towards the managed objects in the managed systems and supervision of notifications transmitted by the managed system.

Allowed operations towards managed objects in the managed system are determined by an information model of the managed system, together with the notifications, which can be transmitted. The information model states:

- which classes of managed objects that are defined,
- which operations the managed objects in these classes accept,
- which notifications that the managed objects are expected to transmit,
- how many instances which can be created or removed, in each class of managed objects,
- dependences between managed objects, e.g. that one managed object requires existence of another one,
- dependences in a managed object, e.g. that a specific attribute value of one attribute is only allowded if another attribute is set to a specific value or if the managed object only can be removed if it is in a specific state,
- the purpose and intention with managed objects and their notifications.

To be meaningful the managing system must know the information model of the managed system. This is in TMN called "shared management knowledge".

At a change of the information model the management model must be updated in accordance with this change. In conventional systems these changes are made by:

- Definition of the new information model. This is made by specifiations for managed objects written as GDMO/ASN.1 templates (GDMO - Guidelines for the definition of Managed Objects according to CCITT rec. X.722 ISO/IEC 10165-4) and ER-diagrams (Entity-Relationship diagrams) for the managed objects. The specifications for the managed objects state formally (machine readable) syntax (e.g. operations and notifications) for the managed object.

All other parts of the information model, as dependences, the number of instances etc., are stated informally as comments in natural language.

- Implementation and verification of the new information model in the managing system and the managed system.

- Confirmation of that the managing system and the managed systems are adapted to the same information model by performing accepted test sequences.

- Updating of the network consisting of the managing system and the managed system with this new version of the information model.

That just mentioned above results in a number of problems:

Firstly, the developement of managing systems and managed systems must be coordinated, which leads to higher developement costs and/or a delayed introduction of new services on the market.

Secondly, the abscence of formalism with respect to the specifications of the managed systems, makes implementation, verification and acceptance of both managing system and managed systems to a difficult and time demanding task, since the interpretation of the specifications is open to discussion.

Thirdly, updating of networks must be planned and carried out carefully, as there are dependences between different versions of the managing systems and managed systems. This involves a delayed introduction of new services in the network.

The purpose of management according to the TMN model is to be a framework for standardisation of management for telecom networks and open systems. The management structure

strongly influences the management paradigm for new system architectures. There are strong reasons to assume the management paradigm according to the TMN model for the whole management and not only for domains subjected to standardisation. The main reason for this is that it is desirable to be able to develop and design management functions in a uniform way.

Summary of the invention

Generally, a first aim of the present invention is to provide a new architecture for management functions, admitting

- cost effective design of management functions,
- less need for coordination and planning of updates of management functions and resources,
- effective support of complex processes, such as network management,
- that the management technology can be developed separately in relation to the resource technology,
- that competence of different kinds can be utilized optimally, i.e. an expert on e.g. the resource domain should focus his efforts to this domain, whereas a person knowing how to design mangement, should be busy with this.

According to a first aspect of the invention this aim, as well as other aims which will appear from the descrition given below, is attained in that the management network defined by way of introduction besides the managed system comprises a generic manager and a representation of the management information model, where the performance of the generic manager during operation is determined by this model representation.

According to a second aspect said aims are attained in that, in the management network in question, there is implemented in the managed system a representation of the management information model as instances of a special class of managed objects.

According to a third aspect said aims are attained in that the management network in question is characterized by

the management information model specification being in the form of a decidable representation of the management in-

formation model, said model defining

- which states of interest from a management point of view the managed system can assume,

- which operations that can be accepted by the managed system,

- which operations that can be directed towards a managed system in a specific state, and finally

- which state the managed system achieves when it is subjected to a specific operation,

wherein with a decidable representation model is meant that the above definitions should be able to be expressed in a machine interpretable language, so as to allow the above mentioned properties to be determinabed from the specification, and,

for defining the state of a certain managed system defining:

- instances of managed objects able to exist,

- attributes they may have and

- possible values of these attributes.

According to a fourth aspect said aims are attained in that the management network in question is characterized by a generic manager, which includes functions which enable an external user to interact with the managed system by manipulating representations of the management information model of the managed system.

Said aims are also attained in that the method defined by way of introduction is characterized in that the managed objects are implemented in the subsystem, uncoordinatedly with respect to other subsystems, in such a way that they can be connected to and transmit messages to other objects in other subsystems, and without knowing the type of objects in the other subsystems.

Various advantagous embodiments of the invention have obtained the characterizing features stated in the respective dependent claims.

By the separation of the managing system the following advantages are achieved:

- the generic manager can be reused for more management functions for more applications in the managed system;

- the generic manager is not influenced by changes in the information model;

- heterogenous networks with a managed system in different versions and with different functions can be managed by one and the same generic manager.

The advantages of the implementation of a representation of the information model in the managed system is the following:

- The representation is always stored in a node in the network.

- The representation always is in accordance with the model of the managed system.

- Easy management of the shared management knowledge.

- Easy management of updates of the system and network.

- Updates of system and network can be made without operation disturbances. No timeslots will appear in which the managing system and the managed system have different opinions with respect to which information model that is valid.

A number of advantages follow from making a specification of the information model in the form of a decidable representation of the information model.

- The generic manager can be made more powerful since it can predict the new state of the managed system after a specific operation, and it can also suggest which operations that are allowed in a specific state.

- The implementation and verification of the managed system as well as management applications are simplified as the expected performance is well specified. Generation of a large part of the implementation code from the specification is also enabled.

- The robustness during operation is improved as only operations leading to allowed state transitions in the managed system can be accepted.

- Early emulation and evaluation of the information model is simplified, which makes the specification task simpler.

- The management model can be formed to be more robust and easy to use by admitting a rather free sequence of operations but still guaranteeing a complete configuration of

the managed system, before the configuration is put into an operational state.

Description of the drawings.

A number of embodiments of the invention will now be described below in greater detail with reference to the enclosed drawings, on which

Figure 1 illustrates in a block diagram one of the basic principles of the invention,

Figure 2 is a flow chart of the imlementation of one of the block units in Figure 1,

Figure 3 is a block diagram of the architecture of the management network according to the invention,

Figure 4 is a block diagram of one of the units in Figure 3,

Figure 5 is a view intended to illustrate a controlling managing system view of one resource,

Figure 6 is a view illustrating the relation between type and instance level,

Figure 7 and 8 are views which illustrate how several units on an instance level share a common unit on a type level in a first and second system, respectively,

Figure 9 is a view which illustrates a controlling managing system view of both systems according to Figure 7 and 8,

Figure 10 is a view which illustrates an example of an information model, which is stored in a type unit,

Figure 11 describes an operation which the controlling managing system can perform by interpretation of the resource representation in Figure 10,

Figure 12 illustrates a change of the information model,

Figure 13 illustrates an example of an operation which violates the information models in fig 10 as well as fig 12, and which never would be directed towards the managed system,

Figure 14 in a block diagram illustrates how old and new technology can coexist in a management domain with two controlling managing systems,

Figure 15 in a block diagram shows a managing system unit comprised in a framework, which is obtained by the invention principle illustrated in Figure 1,

Figure 16 is a view, which illustrates cooperation between a human user and a representation of a resource,

Figure 17 in a similar view as in fig 16 illustrates such an invalid cooperation,

Figure 18 illustrates how resource representations are organised in a data structure,

Figure 19 for the purpose of illustration very schematically shows the general design of a model driven system consisting of a managing system and a managed system,

Figure 20 schematically shows two object types of different classes,

Figure 21 in a similar view as in fig 20 shows the properties of a special object type used when practising the invention,

Figure 22 illustrates schematically the connection between "normal" object classes and objects of the special object type for the formation of a management information model,

Figure 23 illustrates different moments during code generating in connection with the creation of the information model,

Figures 24-29 illustrate moments for enabling interpretation of the information from a special object class in the managed system,

Figures 30-40 are definitions in pseudo-code, where

Figure 30 is a definition of a managed object,

Figure 31 shows how potential attribute values are determined by the attribute type,

Figure 32 is a definition of operations,

Figure 33 states examples of preconditions,

Figure 34 states examples of end conditions,

Figure 35 states examples of end conditions where a managed system checks the consistency,

Figure 36 states examples of end conditions where the managed system keeps the consistency,

Figure 37 is a defintion of an object called LineDriver,

Figure 38 states an example of a dependency scheme,

Figure 39 states an example of an end condition, which is connected to a method,

Figure 40 states an example of an end condition, which is conected to a create-operation,

Figures 41-44 in block and functional diagrams recapitulate applicable parts of what has earlier been described with respect to the invention and its various aspects, particularly with reference to Figure 3 and 4,

Figure 45 gives examples of syntax for the specification of types of managed objects with attributes, methods and pre and end conditions,

Figures 46 and 47 specify with the use of the same syntax two different object types,

Figure 48 likewise with the same syntax specifies dependences between the object types according to Figure 46 and 47,

Figure 49 shows in a block diagram form a concept model, which illustrates relations between end/preconditions and the concepts on which they are applicable,

Figure 50 gives examples with the use of the same syntax as in Figures 45-47 of a specification of implementation mechanisms to keep dependences between object types according to Figure 46 and 47,

Figure 51 likewise with the same syntax specifies an attribute in the object type according to Figure 46 as an attribute derived from an attribute in the related object type according to Figure 47,

Figure 52 specifies the propagation from the one attribute to the other one in Figure 51,

Figure 53 shows in a flow chart form the steps of a transaction with an incorporated consistency check,

Figure 54 shows the public part of a declaration file, which is obtained at compilation of the object types according to Figure 46 to a class in C++ code,

Figure 55 shows likewise in C++ the implementation of a method, which is incorporated in the file according to Figure 54,

Figure 56 shows in C++ the declaration file, which is generated from the specifications according to Figure 47 and 50,

Figure 57 shows in C++ the implementation of two methods

of the object in the declaration file according to Figure 56,

Figure 58 shows in a flow chart form the algorithm for the execution of operations of a transaction, in which a check of preconditions is included,

Figure 59 shows in C++ an extension of the statement file according to Figure 54 by a method to check the precondition for removal of an object,

Figure 60 shows in C++ implementation of two methods of the object according to Figure 54,

Figures 61-69 in block diagrams illustrate problems, which can appear at the use of technology according to the state of the art in connection with the design of managing systems, where

Figures 61-64 concern problems, which can appear in connection with reuse of library components in the managed objects in managed systems,

Figures 65 and 66 refer to problems, which can appear at the implementation of a managed system in a layered system architecture,

Figures 67-70 illustrate how the problems according to Figures 61-66 can be generally solved by design of a specific type of object, which should be able to co-operate with unknown future managed objects,

Figures 71-74 for the same examples, as described above in connection with Figures 45-52 and with the use of the same pseudo-syntax, specifies design of objects according to Figures 70-74, the object type according to Figure 47 being assumed to belong to a platform system,

Figures 75-80 show the implementation in the programme code C++ of dependences between the designed objects according to Figure 71-74.

Description of embodiments

According to one of the characteristics of the invention a separation is made of the managing system into a generic manager and a representation of an information model of a managed system.

In Figure 1 this is illustratated schematically. The managing system is there denoted by 2. The performance of the generic manager 4 is determined by the representation 6 of

the information model 8. From the model representation 6 the generic manager can in other words make decisions related to which operations, that should be made towards a managed system 10, and can decide which operations, that are required to achieve a desired state of this. The model representation 6 is also used for the correct interpretation och packeting of data from and to, respectively, the managed system 10.

When new resources are introduced in the managed system 10 it is only required to change the model representation 6.

According to another characteristic of the invention a representation of the information model is introduced (stored) in the managed system.

For the specification and implementation of managed objects in the system a machine interpretable language is used. In this language the specifications are written for the the managed objects. These specifications of managed objects form together the specification of the management information model.

This specification is conveyed to a compiler, which generates implementation stub code and intermediate format for the management information model.

The implementation stub code is then possibly refined manually and is compiled to implement the managed system. This implementation is then packeted together with the intermediate format of the management information model to loading packages. This loading package is then loaded into the target system, i.e. the managed system. During this loading process the intermediate format of the management information model is used to create a representation of the management information model.

In the managed system the representation of the management information model is implemented as instances of a specific class of managed objects. This class is below denoted MIM-server (Management Information Model Server). For each class of managed objects an instance is created of the MIM-Server class.

The creation of loading packages appears schematically from Figure 2.

According to Figure 2 a specification 12 of the managed

object is written at a first step 11. At the next step 13 this specification is compiled to an implementation stub code 14 in C++ and to an intermediate format 16 of the object specification. For a detailed description of the language C++ references are made to Margaret A Ellis, Bjarne Stroustrup: "The annotated C++ Reference Manual".

According to step 18 the managed object is implemented in C++. This implementation comprises the created stubcode, which comprises the intermediate format of the specification. In step 20 the C++ implementation is compiled to 22.

At loading of a package 30 in the managed system, a new instance of the MIM-server class is created for each class of managed objects, which is implemented by loading packages.

According to a further characteristic of the invention the management information model specification is performed as a decidable representation of the management information model.

The management information model states

- which states the managed system can assume (states of interest from a management point of view),
- which operations that can be directed towards the managed system,
- which operations that can be directed towards a managed system in a specific state, and finally
- which state the managed system achieves when it is subjected to a specific operation .

With a decidable representation of the management information model we mean here that these definitions should be able to be expressed in a machine interpretable language, leading to the above properties being determinable from the specification.

The block diagram in Figure 3 shows the basic design blocks of the architecture of the management function.

In the managing system, here denoted 40, the generic manager 42 is included with a user interface 44. In this interface all managed objects of the managed system can be inspected and modified.

DCF-Data Communication Function 46, which also has been mentioned by way of introduction, is a protocol

implementation, which carries a management protocol as CMIP or similar. This communication function is not a part of the invention.

In the managed system denoted 48 a generic agent 50 is included, which terminates the management protocol.

A number of resource agents 52 are represented for the classes of managed objects in the managed system. For each class of managed objects there is a resource agent. The termination of the management protocol which is unique for the class of managed objects is terminated in the resource agent.

The MIM-server 54, which likewise has been mentioned above, generally is a resource agent for the class MIM-server of managed objects. The MIM-server is a supplier of the representation of the real management information model of the managed system.

The data base management function 56 (DBMS - Data Base Managing system) is an object oriented data base server. It can store, access and update object structured persistent data.

The auxiliary interface 58 for managed objects (MOSI - Managed Object Service Interface) is a general interface, which is supplied by all resource agents. The operations in this interface are create, erase, write, read and method. The operations in MOSI are always directed towards a specific instance of a specific class of managed objects.

The data management interface 60 (DMI - Data Managing Interface) is the interface towards DBMS. It has operations for creation, erasing, reading and updating of object structured persistent data.

The MOSI-interface specification is given below:

```
#ifndef COOAMOSIBASE_HH
#define COOAMOSIBASE_HH
/* $Id: CooaMosiBase.hh,v 1.6 1992/12/01 07:34:52 euassg
$*/

#include<CooaMosiVersion.hh>
```

```
class DelosBuffer;
class DICOS_DbTransaction;

enum CooaGetMode
{
 Cooa_getSpecified=0,
 Cooa_getFirst=1,
 Cooa_getNext=2
};

typedef unsigned int CooaAttributeID;

typedef unsigned int CooaActionID;

typedef unsigned int CooaMoClass;

typedef DelosBuffer CooaAccessControl;

enum CooaResultValue
{
 Cooa_accessDenied=2,
 Cooa_noSuchAttribute=5,
 Cooa_invalidAttributeValue=6,
 Cooa_noSuchAction=9,
 Cooa_processingFailure=10,
 Cooa_noSuchArgument=14,
 Cooa_invalidArgumentValue=15,
 Cooa_missingAttributeValue=18,
 Cooa_classInstanceConflict=19,
 Cooa_mistypedOperation=21,
 Cooa_invalidOperator=24,
 Cooa_invalidOperation=25,
 Cooa_notReplacable=1000,
 Cooa_noDefault=1001,
 Cooa_notAdded=1002,
 Cooa_notRemoved=1003,
 Cooa_false=1004,
 Cooa_invalidCompareMode=1005,
 Cooa_noIteration=1006,
```

```
 Cooa_noMoreAttributes=1007,
 Cooa_ok=1008,
 Cooa_notReadable=1009,
 Cooa_interfaceViolation=1010,
};

enum CooaSetMode
{
 Cooa_replace=0,
 Cooa_toDefault=1,
 Cooa_addMember=2,
 Cooa_removeMember=3,
 Cooa_initiate=4
};

enum CooaCompareMode
{
 Cooa_equal=0,
 Cooa_greaterOrEqual=1,
 Cooa_lessOrEqual=2,
 Cooa_present=3,
 Cooa_subsetOf=4,
 Cooa_supersetOf=5,
 Cooa_nonNullSetIntersection=6,
 Cooa_initialString=7,
 Cooa_anyString=8,
 Cooa_finalString=9
};

enum CooaOpenMode
{
 Cooa_create=0,
 Cooa_delete=1,
 Cooa_update=2,
 Cooa_read=3
};

class CooaMosiBase : public CooaMosiVersion
{
```

```
        public:
        virtual unsigned int Cooa_version () const { return
    2000; };
        virtual void get (CooaGetMode mode,
                CooaAttributeID& attributeNumber,
                DelosBuffer& attributeValue,
                CooaAccessControl& access,
                CooaResultValue& result,
                DelosBuffer& errorInformation)=0;// pure virtual

        virtual void set (CooaSetMode mode,
                CooaAttributeID attributeNumber,
                DelosBuffer& attributeValue,
                CooaAccessControl& access,
                CooaResultValue& result,
                DelosBuffer& errorInformation)=0;// pure virtual

       virtual void action (DelosBuffer& argument,
                CooaActionID actionNumber,
                DelosBuffer& actionResult,
                CooaAccessControl& access,
                CooaResultValue& result,
                DelosBuffer& errorInformation)=0;// pure virtual

        virtual void compare (CooaCompareMode mode,
                CooaAttributeID attributeNumber,
                DelosBuffer& attributeValue,
                CooaAccessControl& access,
                CooaResultValue& result,
                DelosBuffer& errorInformation)=0;// pure virtual
        virtual void mode (CooaOpenMode openMode,
                CooaMoClass moClass,
                CooaAccessControl& access,
                CooaResultValue& result,
                DelosBuffer& errorInformation)=0;// pure virtual
        virtual CooaAttributeID getPrimaryKey(CooaResultValue&
    result) = 0;

        virtual CooaMosiBase* create (DelosBuffer& primaryKey,
```

```
                DICOS_DbTransaction& trans,
                CooaMoClass moClass,
                CooaAccessControl& access,
                CooaResultValue& result,
                DelosBuffer& errorInformation)= 0; // pure
virtual

     virtual void getCounter (CooaAttributeID& attributeNum-
ber,
                void*& counterObject,
                CooaAccessControl& access,
                CooaResultValue& result,
                DelosBuffer& errorInformation) = 0; // pure
virtual
    };

    #endif
```

The general object manager 42 knows the interface of the MIM-server. It reads data from the MIM-server and confirms which classes of managed objects that are in the managed system.

In the user interface 44 can e.g. a request be made that the general object manager shows all instances of a specific class of managed objects. The general object manager can read the definition of the selected class from the MIM-server 54. It knows how this class is accessed via DCF to the generic agent. It knows also how data from the generic agent should be interpreted.

With reference to Figure 4 the resource agent consists of three parts, viz. a MOSI-subagent 61, data object logic 62 (DOL - Data Object Logic), and complementary logic 64 (CL - Complementary Logic). It further provides the external interface 58, MOSI, and has two internal interfaces, viz. a data object interface 66 (DOI -Data Object Interface) and a complementary logic interface 68 (CLI - Complementary Logic Interface).

Here a general description will now be made in greater detail for the principle described with reference to Figure 1

to seperate the managing system in a generic manager and a representation of the model of the managed system. This separation is also called separation model for the purpose of reference.

For management of resources in a system there must exist knowledge with respect to what should obtain management and how it should be performed. For this there are three main functions required, viz. a controlling part of the managing system, below simply called "controlling manager", a resource representation, and the resource. Figure 5 describes in greater detail the relations between the three functions. A problem is to give the controlling manager a general possibility to manage a specific resource. It is therefore necessary to implement a framework which admits the design of applications, which can manage resources without knowing the internal structure of a resource and its possibilibinds. More exactly, new resources should be able to be added and old should be able to be liquidated or be changed without influencing the controlling manager.

Such a framework can be applied to each domain, where it exists a relation between controlling manager and managed system.

It is essential to state that the resource representation always is expressed as statements on a type level. This implies that all instances of a specific type share the same resource representation. In Figure 6 it may be seen that the units T1-T4 on type level exist on their own and that they can be shared between the units I1-I4 on instance level. In a specific example the scheme according to Figure 7 can be valid. The type subscriber number holds information, which describes how instances should be interpreted (e.g. the format of a subscriber number and which kind of data structure, that is used to hold the telephone number).

This structuring technique admits the use of a generic manager. If the scheme in Figure 7 is valid for a system S1 and the scheme in Figure 8 is valid for a system S2 the same generic manager can be used for management of both system S1 and system S2.

The manager view of the two systems is shown in Figure 9.

It is the same generic manager which is used and it interprets the model representations in system S1 and system S2. The management information model of the subscriber number of the systems S1 and S2 are different.

In Figure 9 the manager directs its operations towards resources which should be managed in the system S1 and the system S2. To get the correct syntax and data format the manager interprets the model representations R1 and R2. From the point of view of a user the management framework is the same, i.e. the same concept, metaphores and tools can be used, through which means are obtained for effective and easy management of resources independent of their real implementation.

What is aimed at is now to implement a framework, which can be used for designing generic managers, which manage physical and/or logical resources. The framework is performed trough a rigorous and formal seperation of the resource representation (the model) and the real resource. The resource representation is interpreted by a software unit called controlling manager.

The manager should not need any specific resource information beforehand, i.e. before it has started to work. This means that the manager unit is capable to be adapted to any possible resource representation, which is expressed in such a way that it is interpretable by the manager. The main advantage is that it is possible to introduce new types of resources without updating of the manager.

As an example it is possible with a given resource, such as a subscriber, to introduce new possibilities in the subscriber, and still be able to manage both the new and the old subscriber representation with the same manager.

As another example one can, at implemenation of a totally new type of resource, introduce this in the manager system and manage the new resource with existing managers.

Here a description in greater detail will now be given for the interfaces illustrated in Figure 5.

Access interface for model

This interface is used by the controlling manager to connect to a resource representation and collect information

from it. This information is used at the performance of the general management operations towards a resource. An example appears from Figure 10. By interpreting the resource representation the manager can e.g. perform the operation in Figure 11 on the subscriber instance I1 and be sure that the syntax and semantics are valid with respect to the model.

It is essential to see that the same manager is capable to manage the updating/change of information in the resource representation in Figure 10, which is shown in Figure 12, without the need for being recompiled or reconfigured.

In none of the cases the operation shown in Figure 13 should need to be transmitted to the resource. Faulty operations will thus not at all be transmitted to the resource.

The interface in question has the following operations.

1 - connect_to_representation

2 - interpret_the representation

Access interface for resource.

This interface is used at the performance of management operations towards the real interface. Operations in this interface are often implementation dependent.

Operations:

1 - perform_operation (open/read/write/search etc.)

2 - transform_event.

Information interface for management ability

This interface states which possibilities a specific manager can use at management of the resource. This interface can limit what the manager can do in the access interface for model representation.

Operations:

1 - check_access

2 - limit_access

Information interface for resource ability

This interface states what ability of the resource that will be exported and be shown in its resource representation. Some forms of ability in the real resource can be multiplexed and be stated as one single ability within the resource representation.

Operations:

1 - export_possibility

2 - multiplex_possibility.

The controlling manager

A controlling manager in the framework in question is a software unit capable of interpreting model representations expressed in a machine interpretable way. These representations state the protocol for possible cooperation between manager and the managed resource. The representation states also data structures which are used to represent a concept in the managed resource. Many controlling managers may use one representation and a controlling manager may use many representations.

The representation

A representation of a resource is in the present context a view of some or all aspects "within" a resource. There can be many views of a resource. In the representation the information is structured in such a way that it is possible to interpret it to make transformations to other representations, which are more suitable for e.g. cooperation with man. The representation can be used to feed more or less intelligent software with necessary information for the performance of management operations.

The main advantage with the separation model described above is that it provides means for introduction and updating of resources and services within a management domain without influencing those units, which manage the resources and the services. Further, the units which manage the real resource can be substituted/updated without influencing the units which are being managed. When new technology is available it can thus be introduced at a time which is suitable for the management domain.

Old and new technology can coexist in the management domain. In e.g. a telecom network can equipment (resource units and management units) be introduced in one or several discreet activities.

In Figure 14 a management domain is shown, which consists of two systems with different versions of the units. The versions differ within each system and between the both systems. In this connection the following is valid:

1 - "Old" managers of the type 1 can manage resources of the types 1, 100 and 200.

2 - "New" managers of the type 100 can manage resources of the types 1, 100 and 200.

3 - There are no managers which have been developed to the type 200.

4 - Resources of all types are managed by managers of all types.

In Figure 14 one has a working domain where the technological developement has occurrerd in an uncoordinated way. What we see is that in each layer (horisontally) modifications have been done without influencing the unit from the manager point of view (vertically).

Here it will now be described how the separation model described above is used in connection with designing controlling management units.

The controlling manager responsible for the management operations towards a resource obtains the necessary knowledge within the separation model framework from a model representation, which includes model information of a resource.

The manager is built on the framework which is provided by the separation model, in which the generic manager in this framework is designed according to Figure 15.

The design includes a number of units in a manager layer 80, the denomination of which are apparent from the Figure, and in a model layer 82 a resource representation.

The unit for adapting external user interfaces, at 80, is used in connection with adapting the manager to an external unit. Examples of this are:

- Window systems. This enables a human user to co-operate with the resources by manipulating reproducible representations, as forms, menues and graphical representations.

- Other computers/computer systems.

- Data bases.

- A unit in the managemnt domain itself. The manager can in other words be used as a controlling and decision making unit.

As an example of a representation, with which a human user co-operates, one can consider Figure 16, which shows how a window is related to a management information model according to Figure 10.

This information is also used in connection with verifying that the input signal is correct according to the model. In the example above the user would not be capable to introduce a telephone number, which is not adapted to the resource representation in Figure 10. The manager can use the knowledge to lead the user or suggest what should be done. The user co-operation which is invalid according to Figure 10, and shown in Figure 12, would not be possible to transform to management operations and be transmitted to the resource. The performance described in Figure 17 constitutes the result of the model interpretation ability of a manager, which uses the separation model frame work.

The real managed resource (mySubDir) is still in a valid state and no attempt has been made to perform an invalid operation.

With the type of manager described here, a number of checks are delegated to the user of the model, i.e. the generic manager, instead of being performed by the resource itself. This implies that a resource is capable to fulfill wishes in various user cases. The user cases will probably be differently timed. In a traditional implementation it would be necessary to predict possible future user cases, while in the present framework it is only needed to to make another model representation.

As an example of a more traditional way a "subscriber number resource" could implement the rules described in Figure 10 as C++ statements in the code representing the resource. The use of the resource would then be limited to a context where a telephone number could only start with 727.

The internal representation unit 86 transforms the model representation to a representation, which is suitable for the use by a controlling manager. Abstract data structures etc. in the model are in other words mapped to data structures, which can be used at the working of the manager.

The internal representation unit creates a structure,

which is used for storing the resource representations. There is at least one such structure for each manager domain. From the beginning the structure is empty but is later filled with resource representations. The structure can be created in a number of ways (primary memory, object data base, etc.).

In Figure 18 it may be seen that the resource representations R1-R5 are organised in a data structure. This structure is used by the the controlling manager to get access to the resource representations.

As can be seen, also the resource representations have relations. Such relations constitute part of the model of the manager domain (a subscriber catalogue number has e.g. a relation to LIC).

The generic manager internal representation of the management information model is used by the manager to keep the consistency of the managed system and to decide about the state of the managed system after certain operations have been performed. The operations can e.g. lead to a consistency violation, but it will be detected before the operations are applied to the managed system.

The knowledge, which is obtained by the interpretation of the model, can be used by the manager in different ways, such as:

- automatic solution of the consistency violations and creation of a set of management operations, which do not violate the model,
- guiding of an interactive user to the correct set of operations,
- performance of "what if" analysis.

The generic manager logical unit 88 controls the activity of other units. It makes also decisions based upon events, which are reported from other units. Events can also be reported from units external to the manager, the events of which are always transformed to representations and structures, which can be managed by the manager.

The external representation unit 90 manages transformation of internal representations in the manager to representations, which can lead to and be understood by a unit, which is connected to the unit "adapting of external

user interfaces" at 80. Examples of such a representation is a structure representing a window, which can be reproduced in an X-Windows device.

The unit "Machine" for management operations, at 94, creates real management operations, which should be guided further to a resource in the manager domain. The operation is created as a result of external or internal (or both) corperations/manipulations of a representation controlled by one of the logical units 88 of the manager. A window representing a form is e.g. a representation, which is controlled by the manager. The window can show user alternatives to the user. Some of the activities may result in one or more management operations being transmitted to the managed resource.

The unit access interface for the managing system domain, at 96, is responsible for transmission of the management operations to the controlled unit. The access interface in question can be adapted to various management protocols.

The unit model interpreter at 92 is responsible for interpretation of the resource representations expressed in the models. The model interpreter reads the resource representations and assigns them to the unit described above for internal representation.

The model interpreter interprets the representation of a resource. The representation is expressed in an agreed format. It is the format of the model representation which is used.

For performing a management operation, e.g. to change a subscriber number, the following steps are executed:

1. A connection to the management domain is established.

2. The user (presumably external to the unit for external representation) adresses the resource to be managed (e.g. a subscriber).

3. The manager logical unit checks via the unit for internal representation if there already is an internal representation of the resource intended for management.

4. If such a representation does not exist, the manager logical unit orders "the machine" for management operations to perform management operations via the access interface for

the management domain for collecting a resource representation from the managed system. (Otherwise everything continues with step 7 below).

5. The collected resource representation is transferred to the unit for internal representation, which transforms it to a format suitable for the manager.

6. The unit for internal representation locates the representation in a structure, which represents the sum of the resource representations in the management domain.

7. If such a representation exists the manager logical unit transforms the internal representation to a suitable external representation using the external representation unit.

8. The external representation is guided further to the interface adaption for external users, where the representation can be used by the user (by manipulating a form, which is shown as an X window).

9. The external user changes the value of the subscriber number attribute (the user inputs e.g. a value into a device designated for this).

10. The model interpreter checks if the inputted value violates anything in the resource representation or in the context where the resource representation is present (this implies that a number of management operations may need to be performed towards the managed system).

11. If a violation is detected the user is notified in a suitable way (suggestion, note etc.).

The main advantage of a manager designed as indicated in Figure 15 and used in the separation model context is that it will be capable to manage resource units in a changing environment without need for updating the management units and resource units coordinated with therewith. When developing management and resource units separately the best available technology can be used.

Here a description will now be made of an embodiment, which relates to how the model representation in the above described separation model can be stored in the managed system. The technical problem is how to always keep a managing system with correct management information model

related to a certain managed system, which can be changing with time.

In Figure 19 a mangement system 100 model driven according to the separation model is shown, which system can manipulate objects in another system. The managing system communicates with the managed system 102 via an agent 104. To be able to manipulate the object 106 the managing system needs information, which describes the managed system. The question is how the manager 108 in the managing system obtains this information.

If the managing system can be made to read management information when need arises the managing system becomes more independent of the managed system. If the only dependences between the managing system and the managed system is the way in which the management information is structured, it will be possible to develop the two systems independently. To solve these problems a new object is introduced in the managed system. As the managing system can read data from each managed system at each time the best way to store data about the managed system is to store it in the managed system itself. To use this new kind of objects creates new problems, such as with respect to how they are structured and how they should be installed in the system, at what time etc.

To store information in a managed system, which describes other objects in the managed system, is a way to let the managing system be capable to read such information at any time. To be able to use a common object to describe all kind of objects it is however required that an analysis is made of what kind of information a mangement system needs for performing its tasks.

When structuring all objects in a managed system it can be found that it is the information which describes the classes of managed objects, which needs to be included in the new object. If one e.g. compares the two managed objects of different classes, which are appear in Figure 20, one will see that both object types have attributes, relations and methods. What differs are the names of the attributes etc., and the number of attributes or relations or methods. With the use of this information one can form a template, which

describes managed objects in general. This template is included in the managed object MIM-server, which has been mentioned before, and which is shown in Figure 21. By the use of the template one can describe each managed object, which can be used in the managed system. If one installs an MIM-server-object including information which describes an object type, which we install simultanelously in the managed system, and the managing system can interpret the MIM-server object type, it will become possible for the managing system to collect information about objects in the managed system at any time.

The limitation of what mangement information the MIM-server-object can include, only depends upon how expressivly a managed system can be specified, as the managed objects in question are automatically generated and are filled by data which originate from the object specification itself.

An example of how the structure of the MIM-server-class could look like is shown below. The example is made so that it should be easily readable, but could be specified in an arbitrary language. Some of the abbreviations need an explanation:

- ADT (AbstractDataType) =abstract data type
- DD (DataDomain) = data type;
- Persistent = should be stored in a data base

```
PERSISTENT ADT Mim IS
    PRIMARY KEY myClassId;
    ATTRIBUTES
    myClassId:  Integer;
    myClassName:    String:="NoName";
    myClassVersion:     String:="1.0";
    myAttributeList:    AttributeArray;
    myActionList:   ActionArray;
    myNoticationList:   NotificationArray;
    myNameBindingList:NameBindingArray;
END ADT Mim;
```

It is found that the list of attributes (myAttributeList) is specified as a field of attributes (AttributeArray). In fact the field is a dynamic field, which means that it has no

predertermined size, but instead grows for each element which is added.

```
TYPE AttributeArrayIS
    ARRAY OF Attribute
END TYPE;
```

The field includes elements the names of which are "Attribute", as each element in the crowd will describe properties of a real attribute in a MO-specification. At a closer investigation of the attribute specification it is found that there are attributes as myName etc., which all describe attribute properties:

```
ADT AttributeIS
    ATTRIBUTES
    myName:          String:="NIL";
    optional:    Boolean:=False;
    myExternId:      IntegerArray;
    myInternId:      Integer:=0;
    myDD:        DD;
END ADT Attribute;

TYPE IntegerArrayIS
    ARRAY OF Integer
END TYPE;
```

When considering myDD it can be found that it is defined as DD, i.e. type. This is further explained in the next specification.

```
ADT DD IS
    ATTRIBUTES
    WhichOne:    WhichOne;
    myIntDD:         ItDD            OPTIONAL
    myRealDD:    RealDD          OPTIONAL;
    myTextDD:    TextDD          OPTIONAL;
    myEnumDD:    EnumDD          OPTIONAL;
    myOctetDD:   OctetDD         OPTIONAL;
    myRangeDD:   RangeDD                 OPTIONAL;
    myArrayDD:   ArrayDD                 OPTIONAL;
    myStructDD:      StructDD        OPTIONAL;
```

```
    myRefDD:        ReferenceDD             OPTIONAL;
    myExtDD:        ExternalDD      OPTIONAL;
END ADT DD;
```

Since an attribute is of a type such as an integer, real, string etc., this information is essential. The problem is that an attribute can not be all types at the same time but only one type, why it is necessary to select some various types and to supply more information. This is done by the type WhichOne:

```
TYPE WhichOne IS ENUM
    IntDD,
    RealDD,
    TextDD,
    EnumDD,
    OctetDD,
    RangeDD,
    ArrayDD,
    StructDD,
    RefDD,
    ExtDD
END TYPE;
```

Regular types of attributes are, of cource, integer, decimal or text string, and they could be specified approximatewly as:

```
ADT TextDD IS
    ATTRIBUTES
    isString: Boolean:= True;
END ADT TextDD;

ADT RealDD IS
    ATTRIBUTES
    is32bit: Boolean:= True;
END ADT RealDD
```

There are furthermore certain other common types, which should not be forgotten, such as octet string and enumerated:

```
ADT OctetDD IS
    ATTRIBUTES
    isOctet: Boolean:= True;
END ADT OctetDD;
```

```
ADT EnumDD IS
    ATTRIBUTES
    myEnumList: EnumElementArray;
END ADT EnumDD;

TYPE EnumElementArray IS
    ARRAY OF EnumElement
END TYPE

ADT EnumElement IS
    ATTRIBUTES
    myName: String;
    myValue: Unsigned;
END ADT EnumElement;
```

There are also other types, which are not as usual, but are anyway necessary to enable complete description of an attribute, such as structural type and field type (struct and array). It is also possible to specifiy own types. Range is one type which could be of this category:

```
ADT ArrayDD IS
    ATTRIBUTES
    myElement:  DD;
    mySize:          Integer OPTIONAL;
END ADT ArrayDD;

ADT StructDD IS
    ATTRIBUTES
    myAttributeList: AttributeArray;
END ADT StructDD;

ADT RangeDD IS
    ATTRIBUTES
    myMin: Integer:= 0;
    myMax: Integer:=1;
END ADT RangeDD;

ADT ExternalDD IS
    ATTRIBUTES
    myClassName:         String:= "NIL";
```

```
END ADT ExternalDD;

ADT ReferenceDD IS
    ATTRIBUTES
    myClassId:  Integer     :=0;
    myClassName:    String  :="NIL";
    myInverse:  Integer     OPTIONAL;
END ADT ReferenceDD;
```

An attribute can also have a default value. To be able to include this information a special "default"-type is specified. This type uses also the type WhichOne to state what default value is concerned.

```
ADT Default IS
    ATTRIBUTES
    whichOne:   WhichOne
    myIntDD:                IntDefault     OPTIONAL;
    myRealDD:   RealDefault     OPTIONAL;
    myTextDD:   TextDefault     OPTIONAL;
    myEnumDD:   EnumDefault     OPTIONAL;
    myOctetDD:  OctetDefault    OPTIONAL;
    myRangeDD:  RangeDefault    OPTIONAL;
END ADT Default;

ADT IntDefault IS
    ATTRIBUTES
    myUnSignedValue; UnsignedInteger OPTIONAL;
    mySignedValue: Integer OPTIONAL;
END ADT IntDefault;

ADT RealDefault IS
    ATTRIBUTES
    my32bitValue: Real  OPTIONAL
END ADT RealDefault;

ADT TextDefault IS
    ATTRIBUTES
    myString: String    OPTIONAL
    myChar: Character OPTIONAL;
END ADT TextDefault;
```

```
ADT EnumDefault IS
    ATTRIBUTES
    myValue: EnumElement;
END ADT EnumDefault;

ADT OctetDefault IS
    ATTRIBUTES
    myValue: OctetString;
END ADT OctetDefault;

ADT RangeDefault IS
    ATTRIBUTES
    myValue: Integer;
END ADT RangeDefault;
```

A managed system can be specified by both attribute, methods, notifications etc. By use of this technique for specification of information everything can suit the MIM-server-class. Below follows a list of various other types of information which should be stored:

```
TYPE ActionArray IS
    ARRAY OF Action
END TYPE;

ADT Action IS
    ATTRIBUTES
    myName:          String       :="NIL";
    myInternId:      Integer      :=0;
    myExternId:      IntegerArray;
    myArguments:     AttributeArray;
    myReturnType:    DD;
END ADT Action;

TYPE NotificationArray IS
    ARRAY OF Notification
END TYPE;

ADT Notification IS
    ATTRIBUTES
    myName:          String       :="NIL";
```

```
    myInternId:     Integer     :=0;
    myExternId:     IntegerArray
    myArguments:    AttributeArray;
END ADT Notification;

TYPE NameBindingArray IS
    ARRAY OF Name Binding
END TYPE;

ADT NameBinding IS
    ATTRIBUTES
    myOwnClassName:             String;
    myOwnClassID:           Integer;
    myChildClassName            String;
    myChildClassID:             Integer;
    myChildRDNAttributeName:            String;
    myChildRDNAttributeID:      Integer;
END ADT NameBinding;
```

All MIM-server-objects together constitute the complete model of the managed system. By means of this type of object it will be possible for the managing system to get information of the managed system piece by piece. It will also be possible to change the managed system (i.e. the model) during operation and let the managing system update its view of the managed system without recompiling, by reading information in the MIM-server-objects, which have been installed in the managed system.

Figure 22 illustrates the relation between the MIM-server-instances and the classes of managed objects. For each class 120 and 122, respectively, of "normal" managed objects there is an instance 124 resp. 126 of a class 128 of "special" managed objects in the managed system 130. These "special" managed objects constitute the representation of the management information model 132 of the managed system.

For updating of the managed system only a new MIM-server-object needs to be installed to reflect the new object type in the managed system. The managing system can collect it in the same way as has been done with all other MIM-server-objects, and there will now be an updated view of the managed

system without recompiling of the code. This makes the managing system very stable and the managed system can be updated often without worrying about updating of the managing system.

As an introduction to a description in greater detail below of an example of an embodiment of MIM server code generation, a short summary of what has been earlier described will here first be given with reference to Figure 2.

The generation of MIM-server- and MO-code is a chain which starts with the specification of MO, compare Figure 23. T first the designer specifies MO by use of a special specification language, and compiles then the code by means of a specific compiler. The compiler creates source codes in a standard programming language such as C or C++. The main target code is of course the executable code for the managed system, but this is also the perfect time for generation of intermediate format of the managed object part of the management information model. To be able to do this the compiler must have knowledge of the structure of the MIM-server-class. The compiler considers the MO-specification and counts the number of attributes, methods, etc. and uses this information to design a MIM-server-instance. All this information is introduced in a "Class-method" in the source code of the managed system and can be freely executed after installation in the managed system.

The use of the described technique garantuees that the model agrees with the object it describes, as the generation of MIM-server-information and of object code originates from the same MO-specification. It makes it also easier for the MO-designer, who has as his duty to generate model information. The highly automatised chain of information generation makes it really not only easy to design managed objects but it can also be done quickly and realiably.

An example of MIM-sever-code-generation is now given here:

```
PERSISTENT ADT MO IS
ATTRIBUTES
time:   Time;
```

```
NrOfLinks:  Integer;
LinkName    String;
LinkId:     String;
METHODS
LockLink (IN aValue);
END ADT MO;

ADT Time IS
ATTRIBUTES
hour:   IntRange(0,23),
minute:     IntRange(0,59);
second:     IntRange(0,59);
END Time;
```

The compiler will create source code for the object. In C++ it could look approximately as according to the following (Observe that the code is not exact but has only been made as an example, in which only declaration files are shown for the class MO):

```
class MO
public:
MO();
LockLink(aValue);
void            set_time();
void            set_NrOfLinks();
void            set_LinkName();
void            set_LinkId();
time            get_time();
int             get_NrOfLinks();
char*   get_LinkName();
char*   get_LinkId();
void            init();
private:
Time            time;
int             NrOfLinks;
char*   LinkName;
char*   LinkId;             };
```

Since time was a complex type it will also be a separate class:

```
class Time
```

```
public:
Time();
void      set_hour();
void      set_minute();
void      set_second();
int       get_hour();
int       get_minute()
int       get_second();
void      init();
private:
int       hour;
int       minute;
int        econd;};
```

Now the MIM-server-information can be generated. The compiler will go through the MO-specification and design the MIM-server-instance piece by piece. The information will be collected under a class method, here denoted mimInit:

```
void
MO::MimInint()
// Build the first attribute "Hour"
IntRangeDD tmpIntRangeDD;
tmpIntRangeDD.nyMin(0);
tmpIntRangeDD.myMax(23);

// Set the choice switch to IntRange
WhichOne tmpWhichOne;
tmpWhichOne(IntRangeDD);

// Set the DataDomain values
DD tmpDD;
tmpDD.whichOne(tmpWhichOne);
tmpDD.myIntRangeDD(tmpIntRangeDD);

// Set the attribute values
Attribute tmpAttribute1;
tmpAttribute1.myName("Hour");
tmpAttribute1.myDD(tmpDD);
.
.
```

```
.
// Build the second attribute "Minute"

IntRangeDD tmpIntRangeDD;
tmpIntRangeDD.myMin(0);
tmpIntRangeDD.myMax(59);
WhichOneType tmpWhichOne;
tmpWhichOne (IntRangeDD);

DD tmpDD;
tmpDD.whichOne (tmpWhichOne);
tmpDD.myIntRangeDD(tmpIntRangeDD);
Attribute tmpAttribute2;
tmpAttribute2.myName("Minute");
tmpAttribute2.myDD(tmpRangeDD);
.
.
// Build the third attribute "Second"

IntRangeDD tmpIntRangeDD;
tmpIntRangeDD.myMin(0);
tmpIntRangeDD.myMax(59);
WhichOneType tmpWhichOne;
tmpWhichOne(IntRangeDD);

DD tmpDD;
tmpDD.whichOne(tmpWhichOne);
tmpDD.myIntRangeDD(tmpIntRangeDD);
AttributeType tmpAttribute3;
tmpAttribute3.myName("Second");
tmpAttribute3.myDD(tmpIntRangeDD);
.
.
```

Up to now only those attributes have been created, of which Time is constituted, why also the time attributes itself now must be created, and assign the other attributes thereto.

```
AttributeArray tmpAttList;
tmpAttList.add(tmpAttribute1);
```

```
tmpAttList.add(tmpAttribute2);
tmpAttList.add(tmpAttribute3);

StructDD tmpStructDD;
tmpStructDD.myAttList(tmpAttList);
WhichOneType tmpWhichOne;
tmpwhichOne(StructDD);
DD tmpDD;
tmpDD.whichOne(tmpWhichOne);
tmpDD.myStructDD(tmpStructDD);

Attribute tmpAttribute4;
tmpAttribute4.myName(myTime);
tmpAttribute4.myExternId("OID");
tmpAttribute4.myInternId(1);
tmpAttribute4.optional(FALSE);
myDD(tmpDD);

// Now myTime is done, the next tribute to do is NrOfLinks.

// Next attribute to build is NrOfLinks

IntDD tmpIntDD;
WhichOneType tmpWhichOne;
tmpwhichOne(IntDD);
DD tmpDD;
tmpDD.whichOne(tmpWhichOne);
tmpDD.myIntDD(tmpIntDD);
Attribute tmpAttribute5;
tmpAttribute5;myName/NrOfLinks);
tmpAttribute5;myExternId("OID");
tmpAttribute5;myInternId(2);
tmpAttribute5;optional(FALSE);
tmpAttribute5;myDD(tmpDD);
.
.
// The compiler keeps building attributes this way
// until all attributes and so forth is created
// After that it creates the MIM instance and
```

```
// assigns the different attributes to it.

// The MIM instance is assigned the same name as the
// classid of the class the information represents
AttributeArray tmpAttList;
tmpAttList.add(tmpAttribute4);
tmpAttList.add(tmpAttribute5);
tmpAttList.add(tmpAttribute6);
tmpAttList.add(tmpAttribute7);

Mim 24337;
2433.set_myClassName(myMO);
2433.set_myClassID(24337);
2433.set_myVersion(1,0);
2433.set_myAttList(tmpAttList);

//
24337.set_myNotificationList(tmpNotificationList);
24337.set_myNameBindingList(tmpNameBindingList);    }
24337.set_myActionList(tmpActionList);              }
//
}
```

The class is now ready to be compiled by an ordinary C++ compiler, whereupon it can be installed in the managed system. The method MimInit is now ready for execution and a MIM-server-instance will be created with all introduced information. As has been described earlier the software is first tested and then installed in the managed system.

To be able to execute a method which makes an instance of the MIM-server-class, it is of course necessary that the MIM-server-class already has been installed in the managed system. The MIM-server-class should thus be the first class which is installed in the managed system.

The method MimInit is a class method and is only intended to be executed once, and is thus no general method which a managing system can execute.

When it has been decided that a managed system should be updated with new classes the software must be carefully tested before use. When everything has been done and the

classes should be cleared the managed system software executes the MimInit method in each class for updating. As each class has its own MimInit method, one instance in each class will be installed in the data base. After each MIM-class has been made an instance a note is transmitted which describes which class which has been installed. The managing system, which is a subscriber to these notes, obtains of course the note and decides what should be done, renewal of old MIM-server-information or disregard of the note.

To have a representation of the model of a managed system in the managed system itself instead of having it coded in the managing system has the advantage that it always is in accordance with the managed system which it describes. Likewise can a mangement system be connected to an arbitrary managed system, read its model representation and be capable to manage it. It is also possible to let the managed system have an MIM-server-class describing the MIM-server-class itself, as it is a class similar to each other class, which is installed in the managed system . This could lead to a manager which first would read the MIM-server-class-instance which describes the MIM-server-class, its version etc., and with this information decides how the MIM-server-class-instances in the system should be interpreted.

Here will now be described how a managing system accesses and interprets model information stored in a managed system.

To be able to transmit data between tne managing system and the managed system and clear the information to its original form it is required that the receiving system either knows exactly what is transmitted and in which sequence, or that the information can identify itself, usually by labels or in other words by identifying elements. What method however which is used, some common information or rules, often called a protocol, is required to do this.

As different data types are coded with different number of bits, or even the same number of bits, it is important to know exactly how many bits which are included in each data type, when the received bit string is interpreted. This is usually a problem in this context and a number of different protocols are available. To select between the different

means to transmit and recreate data it is necessary to take into account that the information can be rather complex, and when once the information is in the receiving system, it must be restructured in a usable way.

The model information should therefore be represented in a structure, which is known by both communicating parties. For coding and decoding of simple data types to bit strings the same coding and decoding system, which is used in the managed system for its internal commnuication, is used in the embodiment here described. This coding and decoding system must be known to both parties of the link to avoid corruption of data.

For transmission of simple data types to communicable bit strings it is fundamental to have a strategy concerning the coding and decoding of these bit strings. The method used here is to let an integer type use 32 bits to represent its value. A string is coded by the use of 32 bits integers representing the number of characters in the string and each character is represented by 8 bits.

For coding of simple data types to communicable bit strings each data type has a shift method, which simply illustrates what happens:

Code	Decode
int>>bitstring	bitstring>>int
float>>bitstring	bitstring>>float
char*>>bitstring	bitstring>>char*
char>>bitstring	bitstring>>char
boolean>>bitstring	bitstring>>boolean
.	.
.	.
.	.

By use of the coding and decoding principles for simple data types it is possible to create coding and decoding methods for more complex data structures. This is done simply by implementing the method as a serie of simple coding or decoding methods:

```
complex                Buffer operator >>(Buffer buf,
complex X)
{              {
```

```
int a;          buf>>X.a;
char*b;             buf>>X.b;
}               }
```

```
Buffer operator<<(Buffer buf,complex X)
{
buf<<X.a;
buf<<X.b;
}
```

By using this technique for designing coding and decoding methods for complex data structures it is now possible to design almost any kind of coding or decoding method.

When a managing system initiates a read operation towards an instance in the managed system it must state which attribute it wants to read. The data in the attribute is coded by use of earlier described methods, is transmitted to the managing system and is then decoded. If the receiving system has the same coding and decoding methods as the transmitting system and the receiver knows which method which is used for coding of this information, it will not be any problem to read complex information.

To enable interpretation of the MIM-information the MIM-server-class coding and decoding routines are incorporated in the manager. By this the information, which is received from the managed system, can be interpreted by an instance being made of the MIM-server-class, and thereupon bits are shifted out from the data buffer into the MIM-server-instance.

The mangement system includes the MIM-server-class in its interpretation part model interpreter in Figure 15, and when it reads the attribute values from an MIM-server-instance in the managed system and receives a buffer with coded data, the data are easily decoded by shifting out the bits from the buffer into an empty local instance of the MIM-server-class. As the MIM-server-class has methods for shifting of data to and from buffers this will become an easy task. The main problem appears at the interpretation of other classes than the MIM-serve. In this case it is necessary to use MIM-server-information for decoding of information.

Further explanation can be collected from the embodiment

which will now be now described below with reference to Figure 24-28.

To interpret attribute values, which are read from a usual MO, the managing system requires MIM-server-information, which describes the MO-class. It starts with the creation of en empty instance of a MIM-server and a handle (UsrMO) for the attribute values read from the usual MO read attribute values.

1. MIM local_MIM_Link;//creates a MIM-serverinstance.

2. UsrMO local_UsrMO_Link;//creates a UsrMO instance.

3. get (MIM,Link,myAttList);reads model information.

The above three steps are illustrated in Figure24.

4. Buffer>>local_MIM_Link.myAttList;//the copy is filled with information.

The managing system can now look into the locally represented MIM-server and investigate the class Link. Next step is to deduce the instance data from the MO-object, which it was interested in. This procedure is somewhat more complicated, but can be effected in the following step.

In Figure 25 it is illustrated how the managing system deduces instance data from link A.

5. get(Link, A,all); read all attribute values.

Now instance data from A is waiting in a list of buffers for decoding. The managing system will now use shift operations to obtain the correct amount of bits and decode these to the correct data type.

By looking into the local MIM-server to decide which data type the first attribute is, the first attribute can be obtained and decoded from the buffer.

Figure 26 illustrates how the local MIM-server, representing the MO-class, is used to build the first local representation of the MO-instance.

6. When the managing system comes to the data type Date, it will find that it is structured and needs to go down in the structure for further investigation. This is illustrated in Figure 27.

7. The interpreter looks into the structure "date" and looks through its list of attributes to search for the attribute types. By use of this information the next buffer

can be interpreted. This is illustrated in Figure 28.

8. The interpretation is executed and the managing system is capable of presenting the attribute values to an external user. Figure 29 shows how the data is interpreted and the next attribute type is read from the local MIM-server representation.

Here will now be described how the management information model discussed earlier above can be made decidable. With this we mean here that it should be possible for a computer programme to read its specification and interpret:

- which states the managed system can take (states which are of interest from the managing system point of view)
- which operations can be directed towards the managed system.
- which operations can be directed towards a managed system in a specific state and finally
- which state the managed system gets into when a specific operation is directed towards it.

To be able to be called decidable the management information model must be able to be expressed in a machine interpretable language. Within conventional technology natural language is most often used to express which operations can be directed towards a managed system and which state the managed system gets into when a specific operation is directed towards it.

By making the specification decidable a number of advantages are achieved:

- A generic manager can be made more powerful, since it can predict the new state of the managed system after a specific operation, and it can also propose such operations which are allowed in a specific state and/or lead to a desired state.
- The implementation and verification of the managed systems as well as (not general) management applications are simplified as the expected performance is well specified. It will also be possible to generate a large part of the implementation code from the specification.
- The robustness under operation is improved as only operations, which lead to allowed state transitions in the

managed systems are accepted.

- Early emulation and evaluation of the management information model is simplified, which makes the specification task simpler.

- The management model can be modelled to be both robust and easy to use, by admitting a rather free sequence of operations but still guaranteeing a complete configuration of the managed system before the configuration is cancelled in an active state.

Here will now first possible states be defined.

The state of a managed system in a specific moment is indicated by which instances of managed objects and which attribute values these instances have in the managed system in this moment.

To define the state a specific managed system can take a definition must be given of:

- which instances of managed objects, which can exist and
- which attributes they have and
- which values these attributes can have.

From the specification given in Figure 30, line 1, it is appearent that there could be managed objects of the class Subscriber. Managed objects of this class have the attributes, lines 3-8:

- Number which states the subscriber number,
- AdmState, which states administrative state,
- OpState, which states operational state,
- UsageState, which states usage state,
- Line, which gives reference to a line driver unit for the subscriber physical terminal.

The possible state space for a managed system is enlarged by this specification. The new state space consists of new potential subscribers with all possible combinations of attribute values.

The potential attribute values are determined by the attribute types according to Figure 31, lines 24, 28, 32, 37, 41.

Here a description of how possible operations are defined now follows.

Operations directed towards the managed system create,

erase, manipulate and inspect instances of managed objects and can thus change the state of the managed system.

The create operation creates a new instance of a stated class of managed objects.

The erase operation cancels managed objects.

The write operation changes the value of an attribute of an instance of a class of managed objects.

The method operation calls a method of an instance for a class of managed objects.

The read operation returns the value of an attribute of an instance.

Read, write, create and erase operations are general and have the same syntax and semantic rules for all classes of managed objects. The create and erase operations are specified as possible, or not possible for a given class of managed objects.

The read operations are always possible for each attribute of a class of managed objects and are thus specified implicitly at the definition of the attribute and its type.

The write operation is always free of choice and is specified as defining an attribute as being able to be updated.

The method operation is an escape route to an operation specific for a class of managed objects. These operations are performed as methods. Each method is defined as a method towards instances of the class of controlled objets. Each method receives a list of parameters as input data and returns a result.

Figure 32 defines operations. On lines 61-64 the following operations are defined:

-Read operations of attribute Line, Number, AdmState, OpState, UsageState, lines 62-63,

- Write operation of attribute Line, line 61,

- The method LockRequest, line 64.

This managed object is defined as not having any erase or crerate operation, line 65.

Here follows now a description of how allowed combinations of operation/state can be specified.

Dependending on which instances of managed objects, which

are in a managed system and their attribute values, there is a unique set of allowed operations in a specific state. One example is that an instance can not be erased if it is in use (which is stated by an attribute). Another example is that it is not allowed to create a tenth instance of some class of managed objects, as there is an implementation dependent limitation, which states that there must only be nine instances.

Generally, there are two ways to specifiy these combinations of operation/state, viz. by preconditions and/or by end conditions.

A precondition states which conditions must be fulfilled in order for an operation to be approved. In the present case it is stated for each operation which state the managed system must be in order to accept the instruction. The precondition constitutes a logical part of the class of managed objects.

In our example with the object class Subscriber, we want to prevent that the attribute Line is emptied (is set to NULL) before the subscriber is deblocked (this is stated by the attribute AdmState=unlocked). As mentioned the Line attribute is a reference to the line drive unit of the subscriber physical terminal. It is desirable that this reference exists when Subscriber is used.

The lines 79 and 80 in Figure 33 show how this can be expressed.

An end condition states which conditions must be fullfilled after an updating transaction. In the present case it is possible, for each class of managed objects, to state which state the instances of the class must have to be in a consistent state. The end conditions are logically a part of a class of managed objects or a group of such classes. The latter is valid for dependences between objects.

In the present example with the object class Subscriber we want to prevent that the attribute Adm state is unblocked if the attribute Line is emptied. We want to secure that the latter reference exists when the subscriber is used. In Figure 34 this is expressed on the lines 106 and 107.

An end condition can be fullfilled in one of two ways.

The managing system updates the managed system in such a way that the end conditions are satisfied. In this case the manager has responsibility to fulfill the condition. If the manager ignores this responsibility it will reject the updating transaction of the managed system. The other way to fullfill the end condition is to let the managed system maintain the limitations by automaticly accomplishing the necessary updatings.

In the first case the strategy of the manager is determined from the management information model specification, from which it is determinable which operations are allowed in the the present state. In the other case follow up operations are automatically performed by the managed system.

In the example according to Figure 34 either of these strategies can be considered. If the attribute Line is emptied and AdmState=unlocked AdmState could be set to locked. The end condition in this example prevents however this second strategy. The first strategy must therefore be selected. The end condition is checked before the updating transaction of the managed system is committed and the managing system may thus take over the responsibility to fullfill the condition. Lines 124, 125 in Figure 35 show how this can be expressed.

Line 134 in Figure 36 shows how the second strategy could be specified so that the managed system maintains its consistency.

Thus far only end conditions which regulate the consistency limitations in an object have been discussed. It must however also be possible to state end conditions which specify the consistency limitations between objects.

In Figure 37 the object LineDriver is specified. This object has a relation to the object subscriber in Figure 33. An end condition which regulates the consistency limitation between the two objects must now be able to be specified. On line 118 in Figure 34 and line 163 in Figure 37 it is specified that the two objects constitute a part of the same dependency scheme LineAndSubscriber. This dependency scheme is shown in Figure 38. The dependency scheme in Figure 38, lines 169-171 state that if Subscriber is locked must Line

Driver also be locked.

Here follows now a description in greater detail how definition of a state obtained after an operation is made.

Part of the solution to the problem to enable the determination of the new state of a managed system after the accomplishment of an operation consists of the use of end conditions, described above, together with the rules for strategies to maintain these conditions.

One problem remains however, viz. how to determine the consequences of methods and create operations. For this purpose the concept end conditions are expanded and somewhat changed by enabling connection of the end conditions to methods and create operations.

In the present example, with the subscriber, a precondition has been defined, which states that AdmState must be locked as a prerequisite for accepting that the manager empties the attribute Line. The manager should be able to know how AdmState should be made locked, as the attribute AdmState is only read. The method LockRequest is, in fact, the method which should be used to achieve the desired effect.

Figure 39 shows how this is specified on the lines 176-178.

The difference between these end conditions and the end conditions, which are bound to the classes of managed objects - individual or in groups - is that the condition always is maintained by the managed system.

In Figure 40 another case is shown where it is specified that after the object is created the attribute Line is not equal to NULL. Since the attribute Line is a reference attribute to another object, it is decidable from the specification that the subscriber object itself - creates or finds in another way - an object LineDriver and lets the attribute Line refer to this object.

Here follows now a description in greater detail of how to decide and implement a determinable management model. In this connection, the language C++ already mentioned in another context will come to use.

A management model includes objects and relations. This

is illustrated by Figure 41, where four object types are shown, which are related to each other either directly or indirectly. In such a model there are often consistency limitations and dependences between the objects. In Figure 41 are e.g. a Trunk object 200 dependent on the state of its related Port object 202 in order to be operable. Thus when an attribute (opState) of the operational state in a Port object is deactivated a Trunk object should also be deactivated. Besides there is a similar dependence between administrative state, attribute admstate in the Trunk and Port objects. Similar dependences can be applicated also on other relations in Figure 41.

To be able to make a management information model decidable it should be possible to clearly state such dependences. Then a managing system can predict the consequences of a specific updating operation in a managed system.

Implementation of such dependences in the different applications, which access the data base, leads to difficulties in maintaining the applications and can not guarantee the consistency of the data base. Therefore mechanisms for maintaining of dependencies and consistency limitations should be implemented in the data base system. The logic to maintain the consistency rules related to information stored in the data base should thus constitute part of the logic of the data base rather than the applications accessing the data base. In this way each rule is specified and implemented in just one place.

By way of introduction a recapitulation will now be made of applicable parts of what has been earlier described with respect to the invention and its various aspects, compare specifically Figure 3 and 4 and the corresponding text, with reference to the block diagrams and functional diagrams and shown in Figures 42-44.

The management information model of a managed system 204 consists of a set of managed objects 206 with attributes, methods, relations and consistency rules. Most of these objects are implemented among other things by DOL-logic (Database Object Logic) and persistent data base objects.

Many managed objects (such as those representing a hardware resource) include also a static process 208, compare Figure 42, for accessing and receiving signals from the physical device the object is representing.

The data base objects can however include data, which are accessed from traffic applications and are not visible in the managing system view of the object. The managed objects are really not implemented as real objects, but rather they constitute just views 210 of real implemented objects. The real objects 206 can be considered as resource objects. A resource object is thus an object which includes functionality for both management and traffic management, compare Figure 42.

Each resource object 206 includes a set of operations, which constitute a management view of the object. The generic agent 212 accesses the management view of the respective object. These management views constitute together the management information model 214. This model 214 is thus a view of the underlying complete information model 216.

Each managed object is of a special type. The object types define common properties for all objects of this type, i.e. attributes, methods, relations and consistency rules. For each object type there is a resource agent 218 (Figure 44) which includes the logic for this object type. Each instance of an object type is represented by a data base object 220, which includes the persistent data values of the instance. For each managed object there is thus a resource agent 218 and a data base object 220 (together with among other things static processes).

This appears from Figure 44. The generic agent 212 accesses the resource agent 218 via the MOSI interface 222 (Managed Object Service Interface). This interface terminates operations defined in the management information model. The resource agent 218 is made an object instance. The resource agent DOL logic 224 accesses the instance data base object via a DMI interface 226 (Database Management Interface).

The solution involves a certain machine interpretable language for specifiying persistent objects with properties - established within the areas of conceptual modelling and

object orientation - as attributes, methods and relations, and includes also the above described pre and end conditions.

In the description below and the explanation in greater detail of the pre and end conditions syntax i.a. are used which do not constitute part of an existing language but are only used here to clarify the principles by means of examples. Here used pseudo syntax appear from the earlier described Figures 30-40 and from Figure 45. The description below repeats partly what has been said above with reference to Figure 30-40, although more generalised.

The example in Figure 45 specifies an object type denominated AnObject, line 1. It includes three persistent attributes: attr1, attr2 and attr3 (i.e. attributes the values of which are stored in a data base). The types of values which these attributes can accept are Atype, Integer and Integer, 4-6, respectively. Further AnObject includes two methods, denominated m1 and m2. These methods have each an argument of the types ArgType and AnotherArgType, respectively, lines 9-10. The method m2 returns a value of the type AreturnType, line 10.

Further AnObject in Figure 45 has a precondition which specifies that the value of attr2 must equal the value of attr3 when the method should be executed, line 13. The end condition states that the value of attr2 always must be greater than or equal to the value of attr3, line 16. This is a limitation which must not be violated when a transaction is performed on the data base.

The object type AnObject in Figure 45 shows simple examples of pre and end conditions. Slightly more complicated examples on end conditions are present when the condition relates to more related object types. This can be illustrated with an example, which inco-operates two object types: ResourceA and ResourceB (dependences between these two objects are largly similar to dependences between Trunk and Port in Figure 41.)

Wtih reference to Figure 46, which shows the specification of ResourceA, this object type includes two state attributes: admState and opState. The attribute admState states if an object is in operation or not, line 23,

while opState on the other side states if the object is operative or not, line 24. There is also a so called reference attribute Bref. Such reference attributes implement data base relations between objects. All instances of ResourceA in the data base will include a reference to its related ResourceB instance in the attribute Bref, line 25.

The precondition in Figure 46 states that for removing an instance of ResorceA, admState must equal locked (which means that the object must not be used), line 32. The end condition in Figure 46 states that an instance of ResourceA can only be in operation (admState = unlocked) if it is related to a resourceB object, line 35.

The specification of resourceB is found in Figure 47. It includes, i.a., a reference attribute, Aref, which constitutes the inverse of Bref in resourceA, line 45. As appears from Figure 47 ResourceB has the same preconditions as ResourceA, line 52.

As has been mentioned above there are end conditions which relate both to ResourceA and ResourceB. These end conditions can be considered as dependences between the two object types. One such dependence relates to a subsystem including several object types.

In Figure 48 a dependence scheme is specified, which specifies end conditions related to ResourceA and ResourceB. The first condition states that an object of type ResourceA can not be "unlocked" if its related ResourceB object is "locked", lines 62 and 63. The second condition states that if the value of opState in a ResourceB object is deactivated the value of opState in the corresponding ResourceA object must not be activated, lines 65 and 66.

In Figure 49 a conception model is shown, which illustrates relations between end/preconditions and those concepts on which they are applicable. Figure 49 should speak for itself without a more detailed explanation of the separate blocks.

The semantics of pre and end conditions must be clear and unambiguous. Here the exact definitions of these concepts will now be given. An end condition is specified in a management information model. The end condition states a static consistency rule, which must not be violated in a

managed system, which is adapted to the management information model. More exactly an end condition is applicable to the data base of the managed system. An end condition relates to object instances and their attribute values, which are stored in the data base, compare Figure 49.

As has been mentioned earlier an end condition can for example state a dependence between attribute values. Another example is the cardinality between attributes and data base relations, i.e. limitations related to the number of values of an attribute. It can also be limitations related to the number of instances of an object type.

A data base which must be adapted to a specific end condition, which states a certain limitation, may thus never pass into a state, which violates this limitation. State transitions of data bases appear when operations update stored information.

Updating operations are performed in transactions. A transaction consists of a sequence of operations. The transaction can however be considered as being atomic, in a way that either all operations are performed or none. This is achieved by commit and back rolling operations. The transaction is only performed if each operation is successful. If anything goes wrong during the transaction it is rolled back (i.e. all operations are unmade).

It must thus be guaranteed that the data base takes a consistent state when a transaction is finished. A transaction treats only copies of the data objects and the real objects are not updated before the transaction is committed. This means that the consistency rules may be violated temporarily during the transaction. When the transaction is to be committed no violation of limitations must, however, occur.

A precondition states a rule which must be fullfilled when a specific operation shall be performed, compare Figure 49. More exactly, the data base must be in a state where the rule is fullfilled before the transaction starts.

Preconditions can e.g. be used to limit the sequence of operations on the data base. Limitations of state transitions can also be expressed.

Here follows now a description in greater detail of how implementation mechanisms can be specified.

As concerns end conditions, there are, as has been described earlier, two alternative principles for implementation. One alternative is consistency checks in transactions, which are executed before the transaction is committed. Only if no end condition is violated is the transaction committed, otherwise it is rolled back.

The other alternative solution is automatic correction measures. This can be performed with triggered operations. More exactly, the operations are triggered at specific occasions, e.g. when a specific attribute has been updated.

The specifications in Figure 50 illustrate how these alternative implementation mechanisms can be specified. The first end condition - related to the attribute admState in respective object - is specified for implementation with consistency checks before the transactions are committed, lines 73-77.

The second end condition is specified to be partly performed with automatic corrections. When deactivated is assigned to opState in ResourceB the operation is propagated to opState in ResourceA. When activated is assigned to opState in ResourceB a consistency check must however be made, lines 79-79.

If a completely automatic implementation of the end conditions in Figure 50 is desired the implementation is somewhat more complicated. The attribute opState in ResourceA can be considered as a deriveable attribute, which depends both of the internal state of resourceA and the state of ResourceB. Figure 51 illustrates how opState in ResourceA can be specified as a derived attribute, lines 92-95. The value of opState is calculated from the values of two other attributes: internalOpState and ResourceBstate. The attribute ResourceBstate is a mapping of opState in ResourceB. Each time the value of opState in ResourceB object is changed it should thus be propagated along to ResourceBstate in the related ResourceA object, lines 117, 118. This can be specified in the dependence scheme of ResourceAandB shown in Figure 52.

Here follows now a description in greater detail of how the mechanisms for maintenance of the end and preconditions can be implemented in C++. The implementation can be generated automatically from the above just described specifications for implementation mechanisms.

First the end conditions will be treated.

The implementation of consistency checks is compiled in the respective object type. This means that each object is capable to check all limitations related to objects. The consistency checks of an object should be performed each time a transaction has manipulated the object, before the transaction is committed.

In Figure 53 the steps of the transaction are shown. First all operations are performed according to 250. Then all objects which have been manipulated must perform a consistency check, according to 252. If no cosistency limitation is violated the transaction is finished, 254, otherwise it is rolled back, 256. All updating operations must be made before the consistency checks are performed.

For showing the automatically generated implementation of the consistency checks the example in Figure 46 can be used. If no rules for maintenance of a specific end condition is specified the implementation should at default perform a consistency check. The end condition in ResourceA in Figure 46 should thus be implemented with a consistency check. The automatically generated implementation code is shown in C++. The object type ResourceA is thus compiled to a class in C++. The declaration file for this class is shown in Figure 54.

The class in Figure 54 is, in fact, an implementation of a resource agent. By calling the static method open (line 134), with a value of the main key as parameter, this resource agent can be instanciated to a data base object. For each persistent attribute in the resource object (admState, opState and Bref), there are both a write and a read method to write and read the values of the attribute in th. data base object, to which the resource agent has been instanciated.

That of interest in Figure 54 in this context is the method in line 137. This is a method called before the

transaction is committed for checking the consistency rules. The implementation of this method is found in Figure 55. When end conditions related to several related objects are concerned (as in Figure 48), the consistency check must be performed in each object.

Triggered operations and propagations will now be described.

To illustrate how the implementation of automatic corrections can be generated we can use the specification in Figure 50. Line 82 states that the value of opState in ResourceB should be propagated to opState in ResourceA when deactivated is assigned to opState. The declaration file of ResourceB is found in Figure 56. That mainly of interest are the methods setOpState and propagateOpState, lines 198 and 203 respectively. Figure 57 shows the implementation of these methods.

As appears from Figure 57 the method propagateOpState in the method setOpState is called when the attribute opState is changed from activated to deactivated. This implies that propagateOpState is triggered at state transitions from activated to deactivated. This triggered operation transfers the new value of OpState to the related ResourceA object.

A precondition is a condition which must ve valid when a specific operation should be performed. Opposite to end conditions preconditions must be determined before the operation in question is performed in the transaction. Preconditions can thus hardly be maintained with automatic correction measures, and thus they can only be implemented with consistency checks. Another difference in relation to end conditions is that preconditions must access the original attribute values stored in the data base, instead of the copy in the transaction.

In Figure 58 the algorithm at the performance of the operations in a transaction is illustrated. As appears from Figure 58 at 260, the transaction is rolled back when a precondition is violated.

In Figure 59 a declaration file is shown generated from the specification in Figure 46. The specification in Figure 59 includes a method for evaluation of the precondition

related to the deleteObject operation (line 268). This method should be triggered when the deleteObject operation is to be performed.

Figure 60 shows the automatically generated implementation of the methods deleteObject and deleteObjectCondition in ResourceA. In deleteObject the method deleteObjectCondition is called before the object really is removed. The method deleteObjectCondition reads the value of admState stored in the data base with the method admStateDatabaseValue. If the condition is that admState must be locked an exception is thrown. When the exception is catched the transaction is rolled back. If, on the other hand, the condition is valid the object instance is erased from the data base by the method reallyDelete, which is called in deleteObject.

The advantages of described above are the following.

The implementation code can be generated automatically (compiled) from the same clarifying specification, which is interpreted by the managing system. Modifications of the specification of the model will automatically update both the view of the operator of the information base and the implementation of the same. The managed system can therefore be guaranteed to perform in the way predicted by the managing system.

Each consistency limitation needs only be specified once. The implementation is encapsulated together with the attributes included in the consistency limitation. No consistency checks need to be implemented in those applications which access the data. The maintenance of the code is therefore greatly simplified.

All consistency limitations in a system are implemented in a uniform way. Reliable code is generated automatically.

The implementation strategy is decentralised in such a way that dependences are kept by communication between the included objects. There is no centralised functionality, which must be modified at the introduction of new object types - with new dependences - to a managed system.

The objects included in a transaction can temporarily - before the performance of the consistency checks - be in an

inconsistent state. This simplifies the management of the data base in such a way that the order of operations in a transaction is of less importance.

The solution described here facilitates the implementation of dependences between system components, which are developed uncoordinately. It is even possible to apply the solution on system components compiled and loaded separately.

Here a description in greater detail will now be given of the case uncoordinated implementation of system components.

A managed system consists of several subsystems, more exactly a system platform and a number of applications. These subsystems are developed uncoordinately and are loaded separately.

At the developement of an application - or a system platform - there are libraries with reuasable components. These components should be incorporated and combined in different ways in the different subsystems.

The two different problems described above have certain properties in common. There are system components, which are developed uncoordinately, but still there are dependences between the components. To keep such dependences it is necessary that the components should be able to communicate with other components which are separately developed and even loaded.

The design process described below in greater detail is useable in two differernt contexts with similar problems.

The first problem refers to reuse of library components.

At design of a managed system there are libraries with reusable components, which can be incorporated in the objects of the managed system. As is illustrated in Figure 61 there are first a main library 260 including components with basic functionality, such as MOstateComponent 262 and WorkingStateComponent 264. These components include state attributes common to many different types of managed objects. The functionality of these basic components is reused by other components with a more specialized functionality. In the traffic managemenent library 266 there are e.g. a StatePropagationComponent 268, which has the ability to transfer state transitions in MOstateComponent 262 to the

related dependent object 270.

It should emphasized that components as well as the managed objects are supposed to be implemented in an object oriented language. Both the components and the managed objects are thus really objects. But the components can not exist as identifiable objects by themselves in an application. They can only form a part of managed objects.

A straightforward way to implement components, which depend on other basic components, e.g. the components in the fault handling and the traffic management libraries in Figure 61, is to include the basic components either by heritage or aggregating. This causes however many problems. This depends upon the fact that several components, e.g. FaultHandlingComponent 272 and ResourceManagementComponent 274, which inherit or aggregate the same basic component, e.g. MOstateComponent 262, can be included in the same managed object Route 276, compare Figure 62. This implies that there will be several copies of the basic component in the managed object, which will cause consistency problems if the basic component includes data, which is visible external to the components. Data of MOstateComponent should in other words be available for another functionality in the managed object than those components in which MOstateComponent is included.

There is a way to avoid the problem described above.

MOstateComponent should not be inherited or aggregated in components which depend on this component. Instead the dependent components should include reference attibutes to MOstateComponent. Thus the latter will be a component of its own in those objects in which it is included. Components which need to access MOstateComponent will then each include a reference to the same instance of the MO-state component, compare Figure 63.

As has been mentioned earlier a component, such as ResourceManagementComponent 274 and FaultHandlingComponent 272, can include a functionality which is triggered by state changes in MOstateComponent 262. The latter must therefore be able to transmit state change messages to other components. One problem is however that the receivers of these messages

are unknown at the design of MOstateComponent. MostateComponent 262 must in some way be able to communicate 280, 282 with an arbitrary later appearing component, compare Figure 64.

The second problem refers to a layered system architecture.

A managed system can be implemented in a layered architecture. More exactly, there are system platforms with basic functions, which can be reused by different applications, compare Figure 65. The system platform 290 i.a. includes different kinds of common telecom services. The application can thus delegate many tasks to the system platform. The system platform 290 and the applications 292 are loaded separately. It must be possible to load an application and link it to the system platform without reloading of the platform.

Both the system platform 290 and the applications 292 include managed objects. The objects 294 and 296 in the platform 290 can e.g. represent resources, as switches, transmission routes, trunks etc. The objects 298,300 and 302,304 in the applications 292.1 resp 292.2 can be related to and co-operate with the resources in the system platform, compare Figure 66. An object in an application can delegate some tasks to an object in the system platform. Objects in the applications may thus depend on related objects in the system platform to be able to work. As the system platform is known at the developement of an application it is no problem to design the objects in the applications for communicating with the objects in the system platform. As has been mentioned earlier the objects in the applications can however be dependent on the objects in the system platform. This implies that objects in the system platform should be able to transfer state changes, as has been described earlier above, to objects in the applications. This implies that objects in the platform should be able to be related to and take initiative to calling objects, which are unknown at the design of the platform system.

Above two cases with similar problems have been described. Here will now follow a description of how these problems

can be solved.

More exactly, the real problem in each of the two cases is to design an object, which should be able to communicate with unknown future objects. It must be possible for the original object to be related to and to co-operate with an unknown object without modification, or even reloading of the original object.

This problem can be solved with the design principle illustrated in Figure 67. The original object 330 is designed to co-operate with an abstract object 332. The abstract object defines an interface consisting of unimplemented methods. These are the methods, which can be called by the original object. At the design of an object 334 designated to co-operate with the original object it must inherit the abstract object 332 and implement its inherited methods. At the co-operation with an unknown type of object the original object 330 will consider this as being said abstract type 332. At call of a method defined in the interface of the abstract object 332 the call will be delegated to implementation in the real object 330 via late binding.

Avoidance of reloading of the original object at loading of the future objects is achieved by dynamic linking.

Figure 68 illustrates how the just described design can be used for the design of basic library components. MOstateComponent 340 is designed to communicate with objects, which inherit an abstract object: MostateSubscriber 342. This abstract object defines methods called by MOstateComponent when a state change occurs. If a component should subscribe to state changes in MOstateComponent, it must inherit MostateSubscriber 342 and implement the answer to the respective state transition notifications. The subscribing component must of course also state itself as a subscriber to MostateComponent.

The principles of design described with reference to Figure 67 described can also be used for designing objects in a system platform to co-operate with application specific objects. This is illustrated in Figure 69, where object PlatformObject1 350 is designed to co-operate with object 352 in application system 354. An object in an application, which

should co-operate with the object Platformobject1 350 must inherit the abstract object Interfaceobject 356. Further there is a data base relationship between PlatformObject 350 and InterfaceObject 356. This implies that the objects which inherit InterfaceObject 356 can set up relations with PlatformObject 350. To make it possible to load applications without reloading of the system platform dynamic linking is used.

Often an object in an application system is state dependent of another object in the system platform. This design makes it possible to implement such state dependences in essentially the same way as for objects which are known to each other when they are implemented, as will be described in greater detail further below.

Dependences between uncoordinated objects can be specified and implemented in essentially the same way as between coordinated objects, which has been described earlier above. Here the same examples will be used to show the design and implementation of dependences between uncoordinated objects. It is however assumed here that the object type ResourceB belongs to a platform system. Thus, with reference to Figure 70, various kinds of objects in various application systems can be related to and co-operate with the object ResourceB 370. The object type ResourceB 370 is related to UserObject 372 via a data base relationship. ResourceA 374 inherits UserObject 372 to be able to be related to ResourceB 370. In the following it will be shown how these objects are specified and implemented. The same pseudo syntax as above will be used.

The specification of the object ResourceB is found in Figure 71. The specification includes two state attributes: admState and opState, a method to allocate instances of ResourceB and a precondition which states that to enable erase of a ResourceB object it must be locked. This is essentially the same as earlier. The only difference is that in this example ResourceB is related to UserObject instead of ResourceA. The relation is established via a reference attribute userRef, line 6.

In Figure 72 the object UserObject is specified. It in-

cludes the state attribute admState. It can however be noted that this attribute is specified as being purely virtual, line 19. This implies that the UserObject in fact will not include the attribute admState. The interface of UserObject will however include unimplemented write and read methods for this atribute. The real implementation of these virtual methods will appear in the respective subtypes of UserObject. Besides there are another attribute resourceBstate, which is assumed to keep the value of opState in the related object ResourceB. Further, the UserObject has a reference attribute Bref, which is used to establish relations with objects of the type resourceB, line 21.

A dependency scheme, which specifies end conditions, including both the object UserObject and the object ResourceB is found in Figure 73. There is an end condition, lines 29-31, which states that admState of UserObject depends on admState in ResourceB in such a way that an object UserObject can not be locked up if the related object ResourceB is locked.

In Figure 73 it is further specified that when a value of opState in an object ResourceB is changed, the new value should be propagated to the related instance of UserObject. One can note the difference between the specification in Figure 73 and the specification in Figure 52. In Figure 73 there is no end condition specified which relates to opState, it is only stated that changes of opState in ResourceB should be propagated to the attribute resourceBstate in UserObject, lines 36 and 37. This implies that all subtypes of UserObject are not necessarily opState dependent on ResourceB, but each subtype of UserObject can manage the opState dependence of ResourceB in its own way.

In Figure 74 the specification of ResourceA is shown. ResourceA is specified as being a subtype of UserObject (line 41). This implies that ResourceA inherits the attributes, methods, conditions and the propagations in UserObject. All the purely virtual specifications in UserObject must be specified and implemented in ResourceA. As appears from Figure 67 the value of opState in ResourceA is derived from the attributes internalOpState and resourceBstate (which is

inherited from UserObject), lines 46-49.

As has been mentioned earlier the implementation is essentially the same as when the objects belong to the same system. The difference is that a certain part of the functionality of the dependent object which uses ResourceB is implemented in UserObject. As earlier the code can automatically be generated from the specifications by a compiler. The declaration file which is generated from the specification of UserObject in Figure 72 is found in Figure 75.

The method checkConsistency in UserObject checks the dependence between the admState atrributes, which are stated in the dependency scheme in Figure 73. The implementation of this method is shown in Figure 76. To check this condition the value of admState in UserObject must be read. This explains why there must be a purely virtual specification of admState in UserObject, even if this attribute is not really included in UserObject.

The declaration file which is generated by the specification of ResourceB in Figure 71 is found in Figure 77. It is almost the same as in Figure 56. The implemantation of the method propagateOpstate is however somewhat different, compare Figure 78. The new value is propagated to the attribute ResourceBstate in the related userObject instance. The admState dependence between UserObject and ResourceB must be checked at updating of the object ResourceB as well as at updating of UserObject instances. Therefore, this condition is implemented in the method checkConsistency in ResourceB.

Figure 79 shows the declaration file, which is generated from the specification of ResourceA. The opState dependence of ResourceB is implemented by derivation of the value of opState from resourceBstate and internalOpState, compare Figure 80.

The advantages of the above described are the following.

Dependences and co-operation between uncoordinated objects can be specified and implemented in the same way as dependences between objects in the same subsystem. This implies that they are visible to and can be interpreted by a managing system.

The above illustrated process for maintenance of dependences between uncoordinated objects in a controlled and visible way facilitates implementation of a determinable management model in a layered architecture.

The degree of reuseability of the library components can be improved by a high degree of flexibility when combinations of components are concerned.

Claims

1. A management network with at least one managing system and at least one managed system for telecom or open systems, in which said managed system includes physical and/or logical resources, which by the managing system are considered and managed as managed objects in the form of data images of the resources, and in which the managing system for its operations directed towards the managed system utilizes an information model of the managed system, which includes a description of all managed objects adapted to the mode of operation of the managing system, characterized by the management network including, besides the managed system, a generic manager and a representation of the management information model, where the performance of the generic manager during operation is determined by this model representation.

2. A management network acording to claim 1 , characterized by the representation of the management information model being introduced into the managed system and being available for the generic manager.

3. A management network according to claims 1 or 2, characterized by the management information model specification being in the form of a decidable representation of the management information model, said model defining

- which states of interest from a management point of view the managed system can assume,
- which operations that can be accepted by the managed system,
- which operations that can be directed towards a managed system in a specific state, and finally
- which state the managed system achieves when it is subjected to a specific operation,

in which by a decidable representation model is meant that the above definitions should be able to be expressed in a machine interpretable language, leading to the above properties being determinable from the specification.

4. A management network according to claim 3, characterized by a unique set of allowed operations in a specific state of said managed system being specified by means of preconditions and/or end conditions, which constitute a logical part of the class managed objects, or a group of such classes, in which a precondition states in which state the managed system must be in order for an operation to be accepted, and an end condition states in which state the managed system should be after an updating transaction.

5. A management network according to claim 4, characterized by end conditions being defined so that they are satisfied according to either of two strategies, namely:

- the managing system updates the managed system in such a way that the end conditions are maintained, in which case a manager has the responsibility for the condition being satisfied, and if the the manager ignores this responsibility, the managed system rejects the updating transaction, or

- the managed system maintains the set limitations by automatically carrying through necessary secondary updates in order to satisfy the end condition.

6. A management network according to claim 5, characterized in that binding of the end conditions can be carried through to methods and create-operations.

7. A management network according to any of claims 4-6, characterized in that end conditions are specified in a management information model of a managed system, the end condition stating a static consistency limitation, which must not be violated in the managed system, the end condition being applicable to a data base in the managed system and relates to object instances and their attribute values, which are stored in the data base.

8. A management network according to any of claims 4-7, characterized in that end conditions may state

dependences between attribute values,

cardinality of attributes and data base relationships, i.e. limitations relating to the number of values of an attribute,

limitations related to the number of instances of an

object type.

9. A management network according to any of claims 4-8, characterized by preconditions stating a limitation of the state of a data base of the managed system, which limitation must be fullfilled before a transaction with a specific operation in the data base starts.

10. A management network according to any of claims 5-9, characterized in that at use of

of the first strategy consistency controls are performed in the transaction before it is committed, the transaction being carried through only if no end condition is violated, otherwise it is rolled back,

of the second strategy automatic correction measures are taken in the transaction before it is committed, as e.g. when a specific attribute has been updated.

11. A management network according to any of claims 1-10, characterized in that for designing reuseable components including functionality, which can be included in a managed object in a managed system, a component having a specific functionality may be designed for generating events, to which components unknown to said component can subscribe.

12. A management network according to claim 11, characterized in that for designing components including attributes, events are generated at change of attribute values.

13. A management network according to any of the preceeding claims, for implementing a managed object in a subsystem of the managed system, by subsystem being meant each part of a managed system, which includes one or more managed objects and data base relationships between managed objects, characterized by the managed object being implemented in the subsystem, uncoordinatedly with respect to the other subsystems, in such a way that it can be connected to and transmit messages to other objects in other subsystems and without knowing the type of the objects in the other subsystems.

14. A management network according to claim 13, characterized in that a first object is designed to co-operate with an abstract object defining an interface consisting of unimplemented methods, which can be called by the first object, and at a later design of a second object unknown to

said first object second object and intended to be able to co-operate with the first object, the second object inherits the abstract object and implements the inherited methods, so that the first object at co-operation with the second object will consider it as being of said abstract type.

15. A management network according to claim 14, characterized in that at call of a method defined in the interface of the abstract object, the call will be delegated to the implementation in the real object by means of late binding.

16. A management network according to claims 4 and 15, characterized by end conditions being specified for a group of classes, where the classes can belong to different subsystems.

17. A management network according to any of claims 14-16, characterized in that reloading of said first object in the managed system at loading of said second object is avoided by means of dynamic linking between the respective subsystems.

18. A management network according to any of claims 14-17, characterized by said first and second objects being located in a first and a second subsystem, respectively.

19. A management network according to claim 18, characterized by the use of dynamic linking to enable loading in the managed system of said second subsystem without reloading of said first subsystem.

20. A management network according to any of claims 1-10, characterized in that the generic manager is able to make decisions from the model representation with respect to which operations that can be made towards a managed system, and to decide which operations that are required to achieve a desired state of therof.

21. A management network according to any of claims 1, 2, 3 or 20, characterized in that the representation of the management information model is used for interpreting and packing data structures from and to a managed system, respectively.

22. A management network according to any of claims 1, 2, 3, 20 or 21, characterized by the management information model and the implementation of managed objects in the system

being made with the same specification in a machine interpretable language, in which the specifications of the managed objects in this language together form the specification for the management information model.

23. A management network according to claim 22, characterized by the specification of a managed object being transferred to a compiler, from which implementation code and an arbitrary intermediate format for representation of the management information model of the managed object is generated.

24. A management network according to claim 23, characterized by the implementation being packed together with the intermediate format to a load package, which is loaded into the managed system, and during this loading process the intermediate format is used to add the management information model of the managed object to the system management information model.

25. A management network according to claim 24, characterized by the representation of the management information model being implemented in the managed system as instances of a special class of managed objects, and an instance of this special class being created for each class of managed objects.

26. A management network according to claim 25, characterized in that for installation of objects of said special class in the managed system an installation method is used which, when executed in the implementation code of managed objects, leads to an instance of said special class being installed.

27. A management network according to any of claims 20-26, characterized by the generic manager having a user interface, which is generated during operation based upon the representation of the information model, and in which all managed-objects of the managed system can be inspected and modified.

28. A management network according to claim 27, characterized by the generic manager during operation managing changes in the management information model of the managed system by transforming a representation of the management information model of the managed system to a representation

internal for the generic manager.

29. A management network according to any of claims 26-28, characterized by the generic manager knowing the interface of said special object class, which constitutes a representation of the management information model.

30. A management network according to any of claims 26-29, characterized in that for each class of managed objects there is a resource agent in which termination of the protocol of the managing system unique for the class of managed objects is performed, said resource agent including three parts, namely

a subagent of an auxiliary interface for managed objects, which is a generic interface provided by all subagents, and the operations of which are create, erase, write, read and method, and in which operations always are directed towards a specific instance of a class of managed objects,

a data object logic, and/or

a complimentary logic,

the resource agent further providing the auxiliary interface for managed objects as external interfaces, and has two internal interfaces, viz. one interface for data objects and one interface for the complimentary logic.

31. A management network according to claim 30, characterized by the resource agent definition including the specification of managed objects, which has been obtained by definition of the properties of the managed objects in a machine interpretable language, and defines the properties of the resource agent constituting the managed object.

32. A management network according to claim 30 or 31, characterized by the implementation of the aidinterface being generated from the specification of the managed object.

33. A management network according to claims 1, 2, 3, or 20, characterized in that the generic manager by means of a representation of the management information model is able to interact with a managed system in such a way that concepts in the management information model are transformed to representations and structures suitable for use by an external user, such as a window managing system, a data base manager etc.

34. A management network according to claim 33, characterized in that the generic manager on software basis creates an internal representation of the management information model of the managed system by interpreting it and presenting it for an external user which in this way can interact with the managed system.

35. A management network according to claim 34, characterized by the generic manager using the internal representation of the management information model of the managed system to maintain the model consistency by analysing the interactive pattern of an external user and taking or suggesting operations towards the managed system.

36. A management network according to claims 34 or 35, characterized in that the generic manager by interpreting the internal representation of the management information model of the managed system is able to conduct/suggest operations according to consistency rules for an interactive user of the generic manager.

37. A management network according to claims 1, 2, 3, or 20, characterized by the generic manager including functions enabling an external user to interact with the managed system by manipulating representations of the management information model of the managed system.

38. A management network according to claim 37, characterized by the generic manager including an external representation unit which transforms representations internal to the generic manager to representations adapted to a user external with respect to the generic manager, e.g. a user of a window managing system, data base manager etc.

39. A management network according to claims 37 or 38, characterized by the generic manager including a model interpreter which can interpret the generic representation of the management information model of the managed system.

40. A management network according to any of claims 37-39, characterized by the generic manager including a function which by interpreting an internal representation of the management information model can create one or several syntactically and semantically correct operations which can be directed towards the managed system.

41. A management network according to any of claims 37-40, characterized by the generic manager including access interfaces towards the management network which are used for receiving events occuring in the managed system and for directing operations towards the same, and for transferring and accessing the representation of the management information model stored in the managed system.

42. A management network according to any of claims 37-41, characterized by the generic manager including access interfaces enabling the same generic representations of the management information model of the managed system to be transmitted over different types of communication networks.

43. A management network according to any of claims 37-42, characterized by the generic manager being implemented according to a model, which includes functionality to enable the use of the representation of the management information model of the managed system and to make transformations to representations suitable for an external user without supplying any extra/new information to the representation of the management information model beyond that specified and stored in the managed system.

44. A management network with at least one managing system and at least one managed system for telecom or open systems, in which said managed system includes physical and/or logical resources, which by the managing system are considered and managed as managed objects in the form of data images of the resources, and in which the managing system for its operations directed towards the managed system utilizes an information model of the managed system, which includes a description of all managed objects adapted to the mode of operation of the managing system, characterized in that a representation of the management information model is implemented in the managed system as instances of a special class of managed objects.

45. A management network according to claim 44, characterized by the management information model specification being in the form of a decidable representation of the management information model, said model defining

- which states of interest from a management point of

view the managed system can assume,

- which operations that can be accepted by the managed system,

- which operations that can be directed towards a managed system in a specific state, and finally

- which state the managed system achieves when it is subjected to a specific operation,

in which by a decidable representation model is meant that the above definitions should be able to be expressed in a machine interpretable language, leading to the above properties being determinable from the specification.

46. A management network according to claim 45, characterized by a unique set of allowed operations in a specific state of said managed system being specified by means of preconditions and/or end conditions, which constitute a logical part of the class managed objects, or a group of such classes, in which a precondition states in which state the managed system must be in order for an operation to be accepted, and an end condition states in which state the managed system should be after an updating transaction.

47. A management network according to claim 46, characterized by end conditions being defined so that they are satisfied according to either of two strategies, namely:

- the managing system updates the managed system in such a way that the end conditions are maintained, in which case a manager has the responsibility for the condition being satisfied, and if the the manager ignores this responsibility, the managed system rejects the updating transaction, or

- the managed system maintains the set limitations by automatically carrying through necessary secondary updates in order to satisfy the end condition.

48. A management network according to claims 46 or 47, characterized in that the end conditions can be bound to methods and create-operations.

49. A management network according to any of claims 44-48, characterized by end conditions being specified in a management information model of a managed system, the end condition stating a static consistency limitation, which must

not be violated in the managed system, and being applicable to a data base in the managed system and relates to object instances and their attribute values, which are being stored in the data base.

50. A management network according to any of claims 44-49, characterized in that end conditions may state

dependences between attribute values,

cardinality of attributes and data base relationships, i.e. limitations relating to the number of values of an attribute,

limitations related to the number of instances of an object type.

51. A management network according to any of claims 44-50, characterized by preconditions stating a limitation of the state of a data base of the managed system, which limitation must be fullfilled before a transaction with a specific operation in the data base starts.

52. A management network according to any of claims 44-51, characterized in that at use of

the first strategy consistency controls are performed in the transaction before commitement thereof, the transaction being carried through only if no end conditions is violated, otherwise it is rolled back, the second strategy, automatic correction measures are taken in the transaction before it is committed, as e.g. when a specific attribute has been updated.

53. A management network according to any of claims 44-52, characterized in that for designing reuseable components including functionality, which can be included in a managed object in a managed system, a component with a specific functionality may be designed for generating events, to which components unknown to said component, can subscribe.

54. A management network according to claim 53, characterized in that for the designing components including attributes are events generated at a change of attribute values.

55. A management network according to any of claims 44-54, characterized by the representation of the management information model being used for interpreting and packing of data structures from and to the managed system, respectively.

56. A management network according to any of claims 44-55, characterized by the management information model and the implementation of managed objects being made with the same specification in a machine interpretable language, the specifications of the managed objects in this language together forming the specification of the management information model.

57. A management network according to claim 56, characterized by the specification of a managed object being transferred to a compiler, from which implementation code and an arbitrary intermediate format for representation of the management information model of the managed object is generated.

58. A management network according to claim 57, characterized by the implementation being packed together with the intermediate format to a load package, which is loaded into the managed system, the intermediate format being used during this loading process for adding the management information model of the managed object to the system management information model.

59. A management network according to claim 58, characterized in that for installation of objects of said special class in the managed system an installation method is executed in the implementation code of managed objects which involves that an instance of said special class is installed.

60. A management network according to claims 44 or 45, characterized in that the generic manager by means of a representation of the management information model may interact with a managed system in such a way that concepts in the management information model are transformed to representations and structures suitable for use by an external user, such as a window managing system, a data base manager etc.

61. A management network according to claim 60, characterized in that the generic manager, on a software basis, creates an internal representation of the management information model of the managed system by interpreting and presenting it for an external user which in this way can interact with the managed system.

62. A management network according to claim 61, characterized by the generic manager using the internal representation of the management information model of the managed system to maintain the model consistency by analysing the interactive pattern of an external user and to take or suggest operations towards the managed system.

63. A management network according to claims 61 or 62, characterized in that the generic manager by interpreting the internal representation of the management information model of the managed system is able to conduct/suggest operations according to consistency rules, for an interactive user of the generic manager.

64. A management network according to claims 44 or 45, characterized by the generic manager including functions which enable an external user to interact with the managed system by manipulating representations of the management information model of the managed system.

65. A management network according to claim 64, characterized by the generic manager including an external representation unit which transforms internal representations internal to the generic manager to representations adapted to a user external with respect to the generic manager, e.g. an external user of a window managing system, a data base manager etc.

66. A management network according to claim 64, characterized by the generic manager including a model interpreter which can interpret the generic representation of the management information model of the managed system.

67. A management network according to any of claims 64-67, characterized by the generic manager including a function which by interpreting an internal representation of the management information model can create one or several syntactically and semantically correct operations which can be directed towards the managed system.

68. A management network according to any of claims 64-67, characterized by the generic manager including access interfaces to the managed system which are used for receiving events in the managed system and for directing operations to the same, and for transferring and accessing the

representation of the management information model stored in the managed system.

69. A management network according to any of claims 64-68, characterized by the generic manager including access interfaces, which enable the same generic representations of the management information model of the managed system to be transmitted over different types of communication networks.

70. A management network according to any of claims 64-69, characterized by the generic manager being implemented according to a model, which includes functionality to enable the use of the representation of the management information model of the managed system and to make transformations to representations suitable for an external user without supplying any extra/new information to the representation of the management information model beyond that specified and stored in the managed system.

71. A method for implementing a managed object in a subsystem of a managed system in a management network with at least one managing system and at least one managed system, for telecom or open systems, wherein which with a subsystem is meant each part of a managed system including one or more managed-objects, characterized in that the managed objects are implemented in the subsystem, uncoordinatedly with respect to other subsystems, in such a way that they can be connected to and transmit messages to other objects in other subsystems, and without knowing the type of objects in the other subsystems.

72. A method according to claim 71, characterized in that a first object is designed for co-operation with an abstract object defining an interface consisting of unimplemented methods, which may be called by the first object, and that at later design of a second object unknown to said first object and intended to co-operate with the first object, the other object inherits the abstract object and implements the inherited methods, so that the first object at co-operation with the second object will consider it as being of said abstract type.

73. A method according to claim 72, characterized in that at call of a method defined in the interface of the abstract

object the call will be delegated to the implementation in the real object by means of late binding.

74. A method according to claims 72 or 73, characterized in that reloading of said first object in the managed system in case of loading of said second object is by means of dynamic linking between the respective subsystems.

75. A method according to any of claims 72-74, characterized by said first and second objects being located in first and second subsystem, respectively.

76. A method according to claim 75, characterized by the use of dynamic linking to enable loading in the managed system of said second subsystem without reloading of said first subsystem.

77. A method according to claim 71, characterized by the generic manager including functions which enable an external user to interact with the managed system by manipulating representations of the management information model of the managed system.

78. A method according to claim 77, characterized by the generic manager including an external representation unit which transforms representations internal to the generic manager to representations adapted to a user external with respect to the generic manager, e.g. a user of a window managing system, a data base manager etc.

79. A method according to claims 77 or 78, characterized by the generic manager including a model interpreter which can interpret the generic representation of the management information model of the managed system.

80. A method according to any of claims 77-79, characterized by the generic manager including a function which by interpreting an internal representation of the management information model can create one or several syntactically and semantically correct operations which can be directed towards the managed system.

81. A method according to any of claims 77-80, characterized by the generic manager including access interfaces towards the management network which are used for receiving events in the managed system and for directing operations towards the same, and for transferring and accessing the

representation of the management information model stored in the managed system.

82. A management network according to any of claims 77-81, characterized by the generic manager including access interfaces, which enable the same generic representations of the management information model of the managed system to be transmitted over different types of communication networks.

83. A management network according to any of claims 77-82, characterized by implementing the generic manager according to a model, which includes functionality to enable the use of the representation of the management information model of the managed system and to make transformations to representations suitable for an external user without supplying any extra/new information to the representation of the management information model beyond that specified and stored in the managed system.

84. A management network with at least one managing system and at least one managed system for telecom or open systems, in which said managed system includes physical and/or logical resources, which by the managing system are considered and managed as managed objects in the form of data images of the resources, and in which the managing system for its operations directed towards the managed system utilizes an information model of the managed system, which includes a description of all managed objects, adapted to the mode of operation of the managing system, characterized by

the management information model specification being in the form of a decidable representation of the management information model, said model defining

- which states of interest from a management point of view the managed system can assume,
- which operations that can be accepted by the managed system,
- which operations that can be directed towards a managed system in a specific state, and finally
- which state the managed system achieves when it is subjected to a specific operation,

wherein with a decidable representation model is meant that the above definitions should be able to be expressed in

a machine interpretable language, so as to allow the above mentioned properties to be determinable from the specification, and,

for defining the state of a certain managed system defining:

- instances of managed objects able to exist,
- attributes they may have and
- possible values of these attributes.

85. A management network according to claim 84, characterized in that a unique set of allowed operations in a specific state of said managed system is specified by means of preconditions and/or end conditions, which constitute a logical part of the class of managed objects, or a group of such classes, where a precondition states in which state the managed system must be in order for an operation to be accepted, and an end condition states in which state the managed system should be after an updating transaction.

86. A management network according to claim 85, characterized in that end conditions are defined so as to being satisfied according to either of two strategies, namely:

- the managing system updates the managed system in such a way that the end conditions are maintained, in which case a manager has the responsibility for the condition being satisfied, and if the the manager ignores this responsibility, the managed system rejects the updating transaction, or
- the managed system maintains the set limitations by automatically carrying through necessary secondary updates in order to satisfy the end condition.

87. A management network according to claim 85 or 86, characterized in that binding of the end conditions may be made to methods and create-operations.

88. A management network according to any of claims 85-87, characterized by end conditions being specified in a management information model of a managed system, in which the end condition states a static consistency limitation, which must not be violated in the managed system, the end condition being applicable to a data base in the managed system and relates to object instances and their attribute

values, which are being stored in the data base.

89. A management network according to any of claims 85-88, characterized by end conditions being able to state

dependences between attribute values,

cardinality of attributes and data base relationships, i.e. limitations relating to the number of values of an attribute,

limitations related to the number of instances of an object type.

90. A management network according to any of claims 85-89, characterized by preconditions stating a limitation of the state of a data base of the managed system, which limitation must be fullfilled before a transaction with a specific operation in the data base starts.

91. A management network according to any of claims 86-90, characterized in that at use

of the first strategy consistency controls are performed in the transaction before it is committed, the transaction being carried through only if no end conditions is violated, otherwise it is rolled back,

of the second strategy, automatic correction measures are taken in the transaction before it is committed, as e.g. when a specific attribute has been updated.

92. A management network according to any of claims 84-91, characterized in that for designing reuseable components which include functionality, which can be included in a managed object in a managed system, a component with a specific functionality may be designed for generating events, to which components unknown for said component, can subscribe.

93. A management network according to claim 92, characterized in that for the designing components including attributes, events are generated in case of a change of attribute values.

94. A management network according to any of claims 84-93, in which a managed object should be implemented in a subsystem of the managed system, by a subsystem being meant each part of a managed system, which includes one or more managed objects and data base relationships between managed objects,

characterized in that the managed object is implemented in the subsystem, uncoordinated relative to the other subsystems, in such a way that it can be connected to and transmit messages to other objects in other subsystems and without knowing the type of the objects in the other subsystems.

95. A management network according to claim 94, characterized in that a first object is designed to co-operate with an abstract object defining an interface consisting of unimplemented methods, which can be called by the first object, and at later design of a second object unknown to said first object intended to be able to co-operate with the first object, the second object inherits the abstract object and implements the inherited methods, so that the first object at co-operation with the second object will consider this as being of said abstract type.

96. A management network according to claim 95, characterized in that at call of a method defined in the interface of the abstract object the call will be delegated to the implementation in the real object by means of late binding.

97. A management network according to claims 84 and 96, characterized by end conditions being specified for a group of classes where the classes can belong to different subsystems.

98. A management network according to claims 95 or 96, characterized reloading of the original object in the managed system at loading of said second object is avoided by dynamic linking between the respective subsystems.

99. A method according to any of claims 96-98, characterized by said first and second objects being located in first and second subsystem, respectively.

100. A management network according to claim 99, characterized by the use of dynamic linking to enable loading in the managed system of said second subsystem without reloading of said first subsystem.

101. A management network according to claim 84, characterized in that the generic manager by means of a representation of the management information model is able to interact with a managed system in such a way that concepts in the

management information model are transformed to representations and structures suitable for use by an external user, such as a window managing system, a data base manager etc.

102. A management network according to claim 101, characterized in that the generic manager, on a software basis, creats an internal representation of the management information model of the managed system by interpreting and presenting it for an external user which in this way can interact with the managed system.

103. A management network according to claim 102, characterized by the generic manager using the internal representation of the management information model of the managed system to maintain the model consistency by analysing the interactive pattern of an external user and to take or suggest operations towards the managed system.

104. A management network according to claims 102 or 103, characterized in that the generic by interpreting the internal representation of the management information model of the managed system is able to conduct/suggest operations according to consistency rules, for an interactive user of the generic manager.

105. A management network according to claim 84, characterized by the generic manager including functions which enable an external user to interact with the managed system by manipulating representations of the management information model of the managed system.

106. A management network according to claim 105, characterized by the generic manager including an external representation unit which for the generic manager transforms internal representations to representations adapted to a user external with to the generic manager, e.g. a user of a window managing system, data base manager etc.

107. A management network according to claims 105 or 106, characterized by the generic manager including a model interpreter which can interpret the generic representation of the management information model of the managed system.

108. A management network according to any of claims 105-107, characterized by the generic manager including a function which by interpreting an internal representation of

the management information model can create one or several syntactically and semantically correct operations which can be directed towards the managed system.

109. A management network according to any of claims 105-108, characterized by the generic manager including access interfaces towards the managed system which are used for receiving events in the managed system and for directing operations towards the same, and for transferring and accessing the representation of the management information model stored in the managed system.

110. A management network according to any of claims 105-109, characterized by the generic manager including access interfaces, which enable the same generic representations of the management information model of the managed system to be transmitted over different types of communication networks.

111. A management network according to any of claims 105-110, characterized by the generic manager being implemented according to a model, which includes functionality to enable the use of the representation of the management information model of the managed system and to make transformations to representations suitable for an external user without supplying any extra/new information to the representation of the management information model beyond that specified and stored in the managed system.

112. A management network with at least one managing system and at least one managed system for telecom or open systems, in which said managed system includes physical and/or logical resources, which by the managing system are considered and managed as managed objects in the form of data images of the resources, and in which the managing system for its operations directed towards the managed system utilizes an information model of the managed system, which includes a description of all managed objects, adapted to the mode of operation of the managing system, characterized by a generic manager, which includes functions which enable an external user to interact with the managed system by manipulating representations of the management information model of the managed system.

113. A management network according to claim 112, charac-

terized by the generic manager including an external representation unit which transforms representations internal to the generic manager to representations adapted to a user external with respect to the generic manager, as a user of e.g. a window managing system, a data base manager etc.

114. A management network according to claims 112 or 113, characterized by the generic manager including a model interpreter which can interpret the generic representation of the management information model of the managed system.

115. A management network according to any of claims 112-114, characterized by the generic manager including a function which by interpreting an internal representation of the management information model can create one or several syntactically and semantically correct operations which can be directed towards the managed system.

116. A management network according to any of claims 112-115, characterized by the generic manager including access interfaces towards the managed system which are used for receiving events in the managed system and for directing operations towards the same, and for transferring and accessing the representation of the management information model stored in the managed system.

117. A management network according to any of claims 112-116, characterized by the generic manager including access interfaces, which enable the same generic representations of the management information model of the managed system to be transmitted over different types of communication networks.

118. A management network according to any of claims 112-117, characterized by the generic manager being implemented according to a model, which includes functionality to enable the use of the representation of the management information model of the managed system and to make transformations to representations suitable for an external user without supplying any extra/new information to the representation of the management information model beyond that specified and stored in the managed system.

Fig. 1

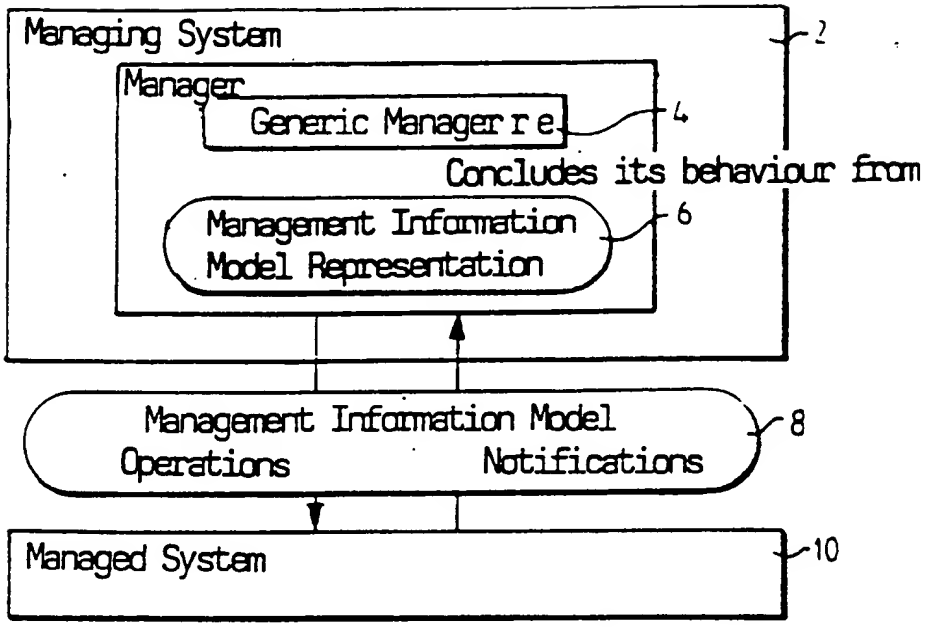

Managing System
2
Manager
Generic Manager r e
4
Concludes its behaviour from
Management Information
Model Representation
6
Management Information Model
Operations
Notifications
8
Managed System
10

Fig. 2

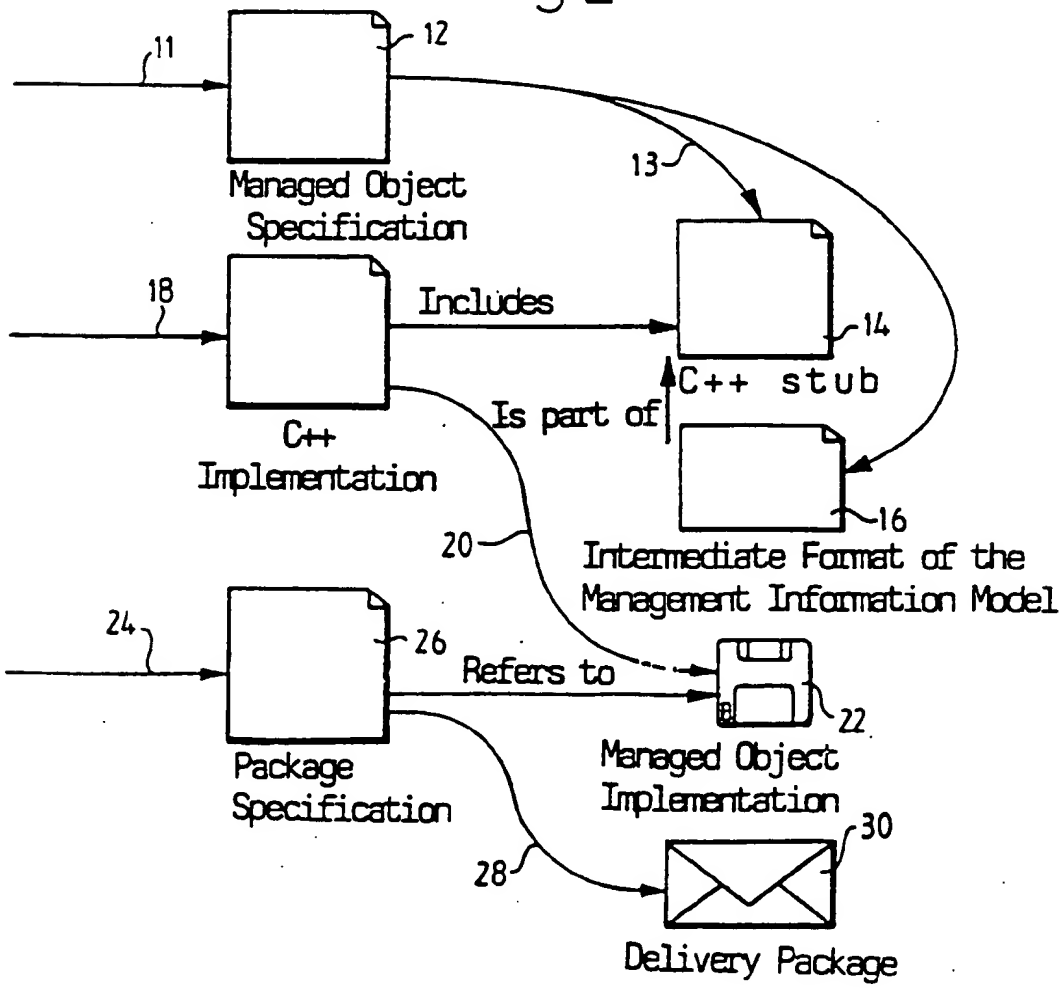

11
12
Managed Object
Specification
13
18
Includes
14
C++ stub
C++
Implementation
Is part of
16
20
Intermediate Format of the
Management Information Model
24
26
Refers to
22
Package
Specification
Managed Object
Implementation
28
30
Delivery Package

Fig. 3

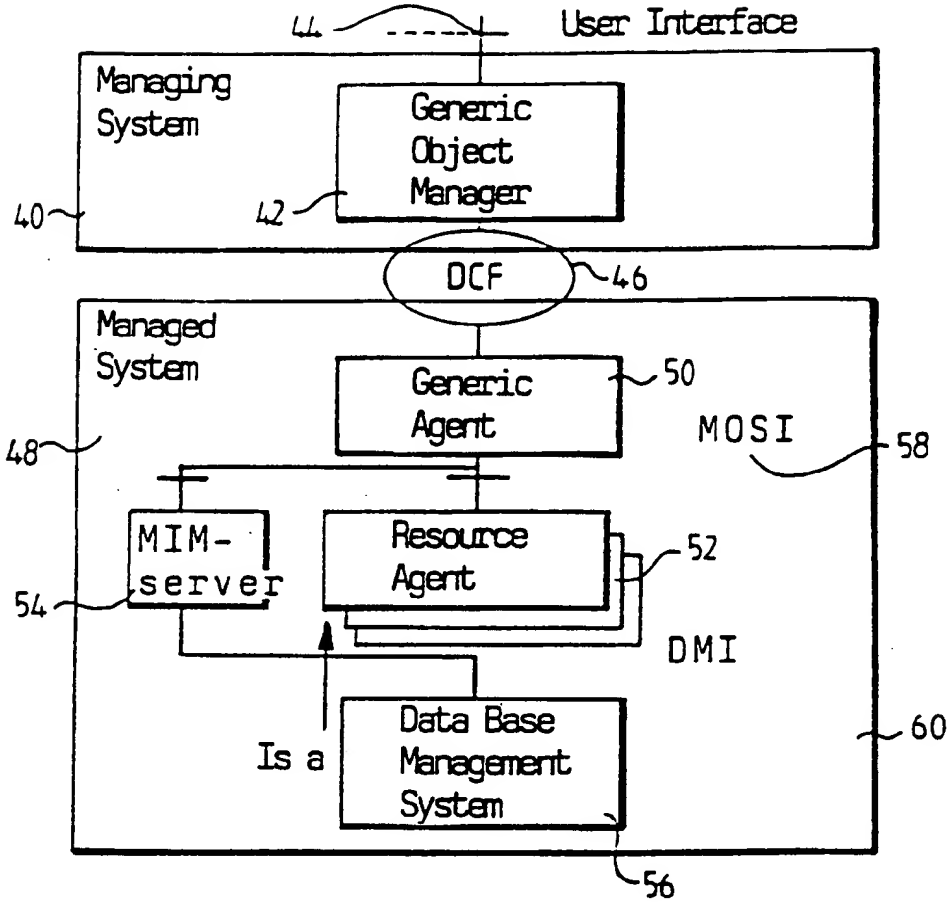

44
User Interface
Managing System
Generic Object Manager
40
42
DCF
46
Managed System
Generic Agent
50
MOSI
58
48
MIM-server
Resource Agent
52
54
DMI
Is a
Data Base Management System
60
56

Fig. 4

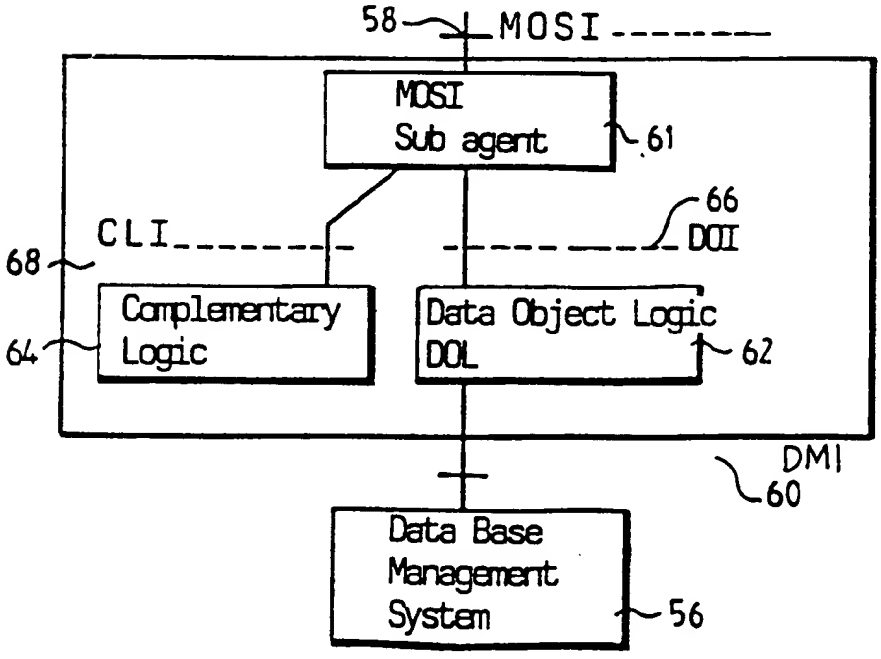

58
MOSI
MOSI Sub agent
61
66
CLI
DOI
68
Complementary Logic
Data Object Logic DOL
64
62
DMI
60
Data Base Management System
56

Fig.5

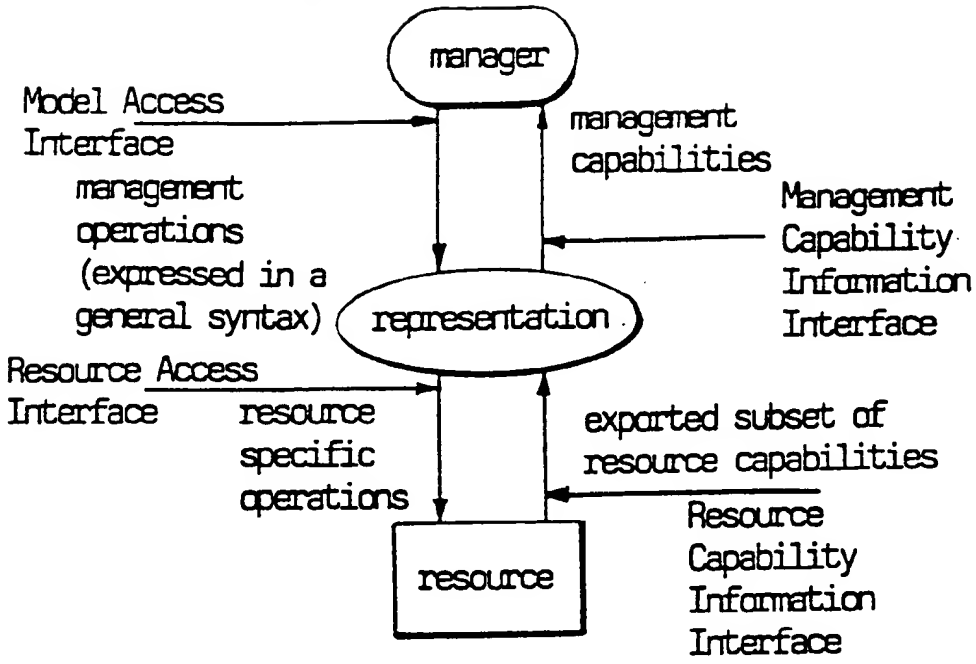


Fig.6

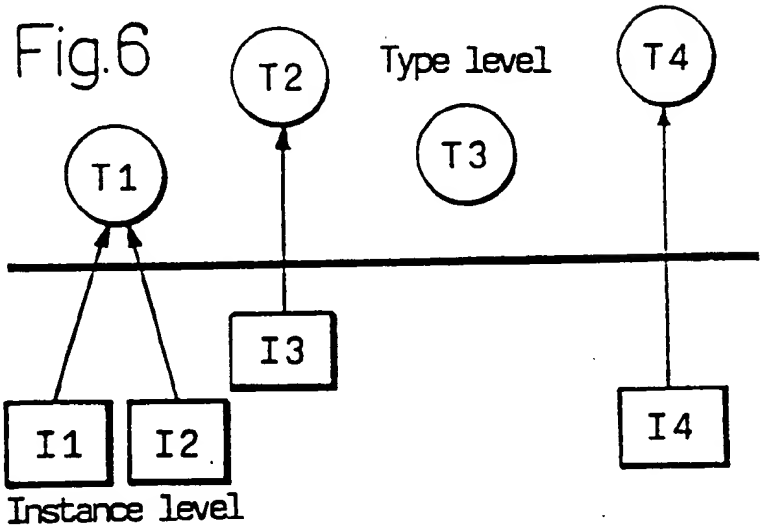


Fig.7

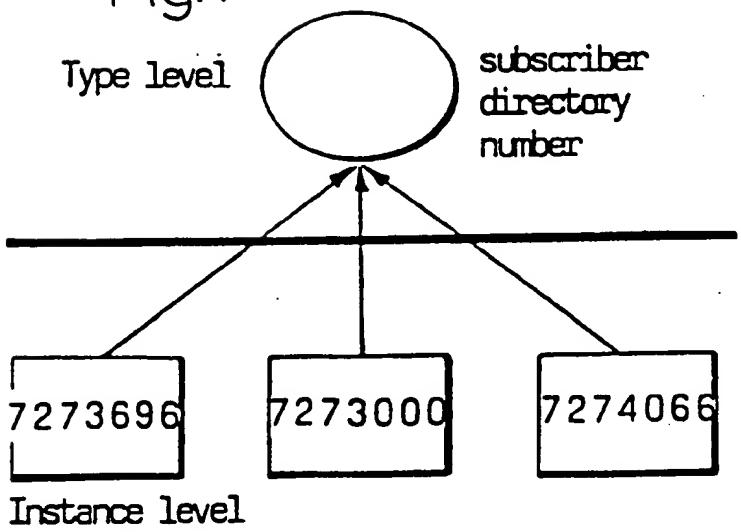


Fig 8

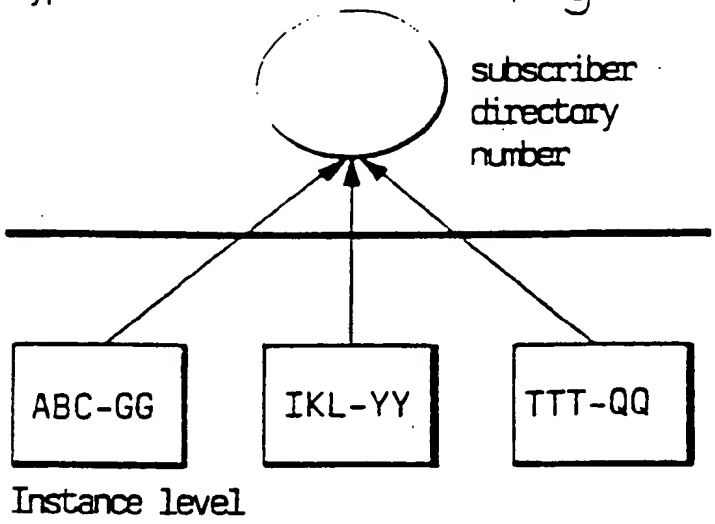


Fig.9

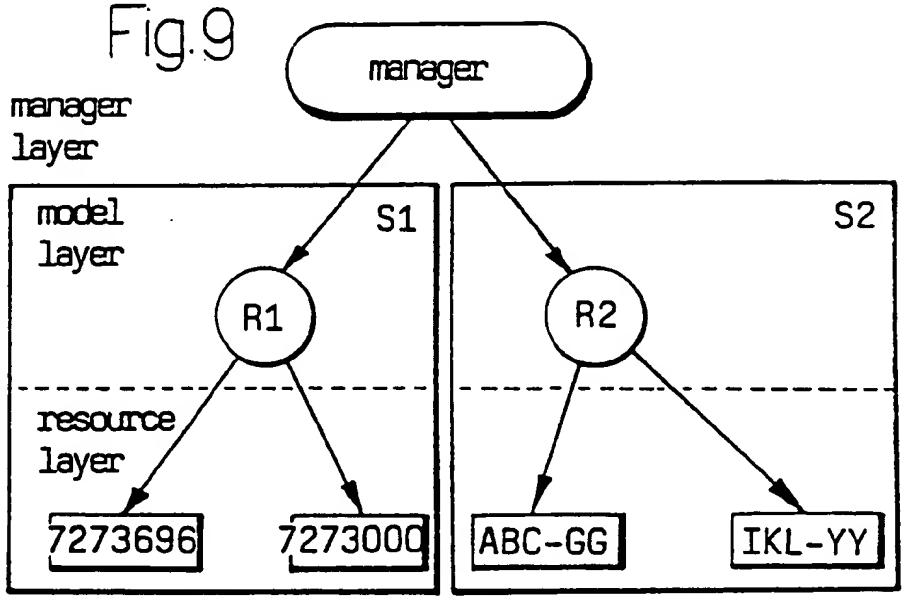


Fig.10

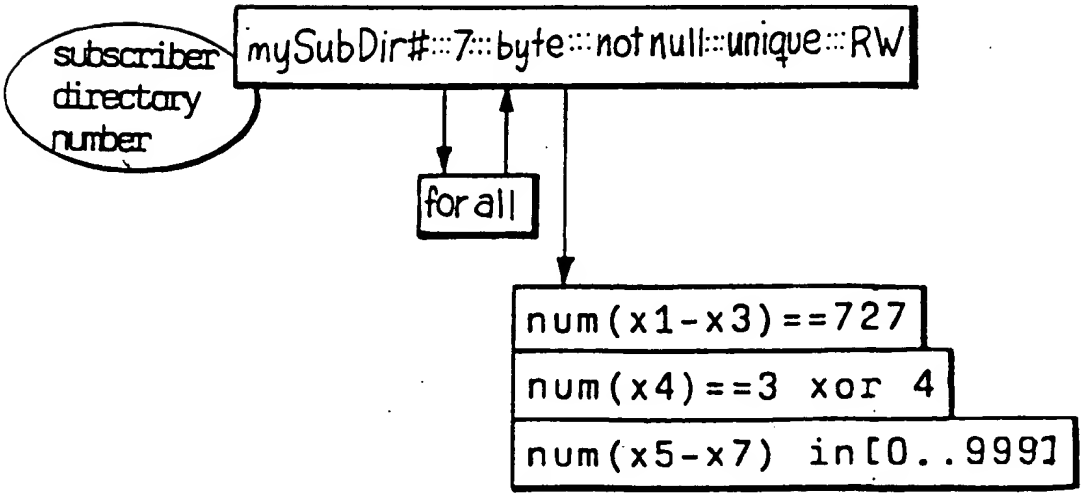


Fig. 11

WRITE (my SubDir#, 7273696

Fig. 12

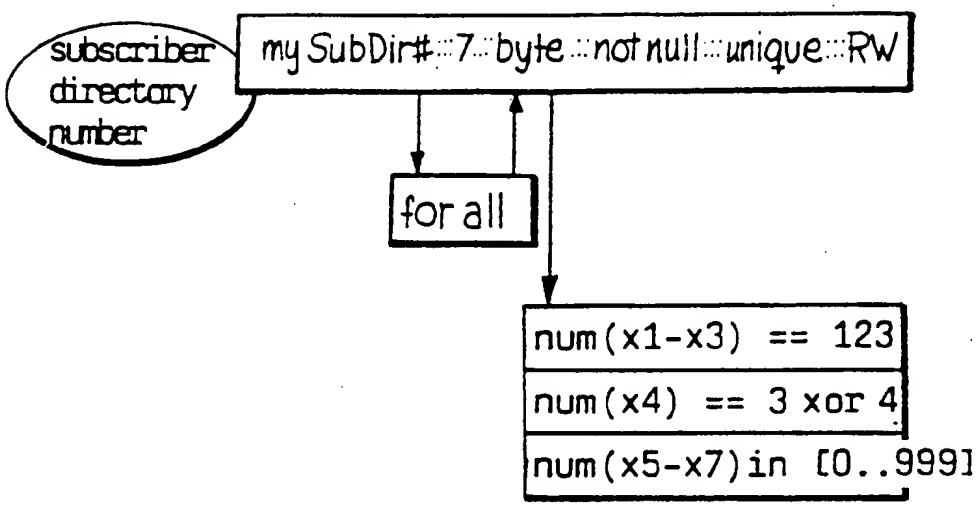


Fig. 13

WRITE (my SubDir# 5553212)

Fig. 14

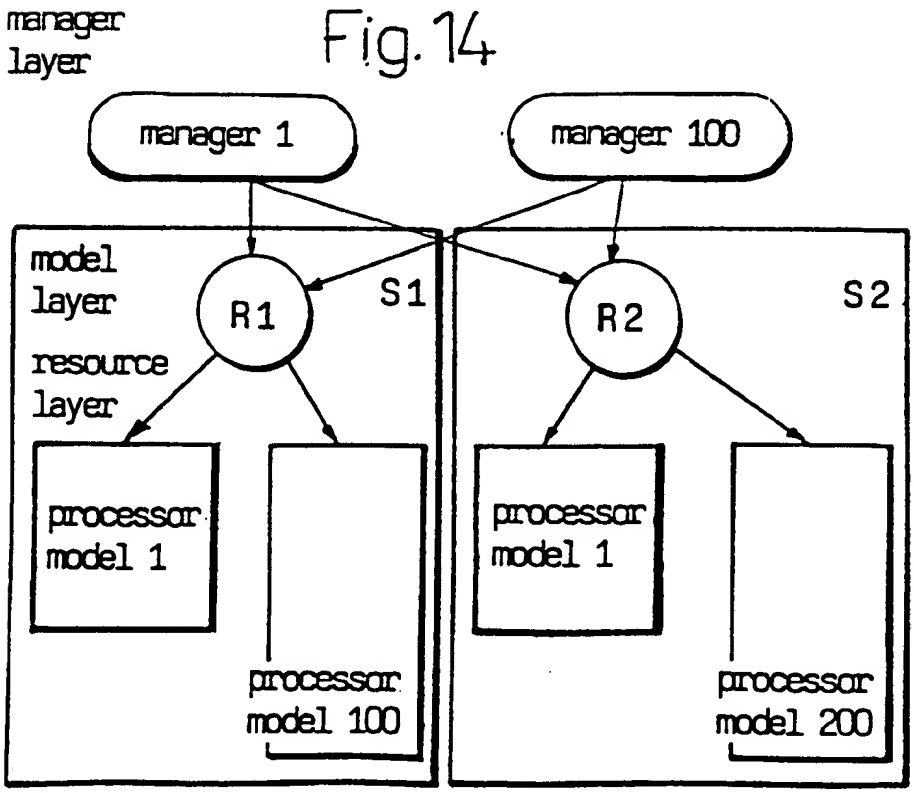


Fig. 15

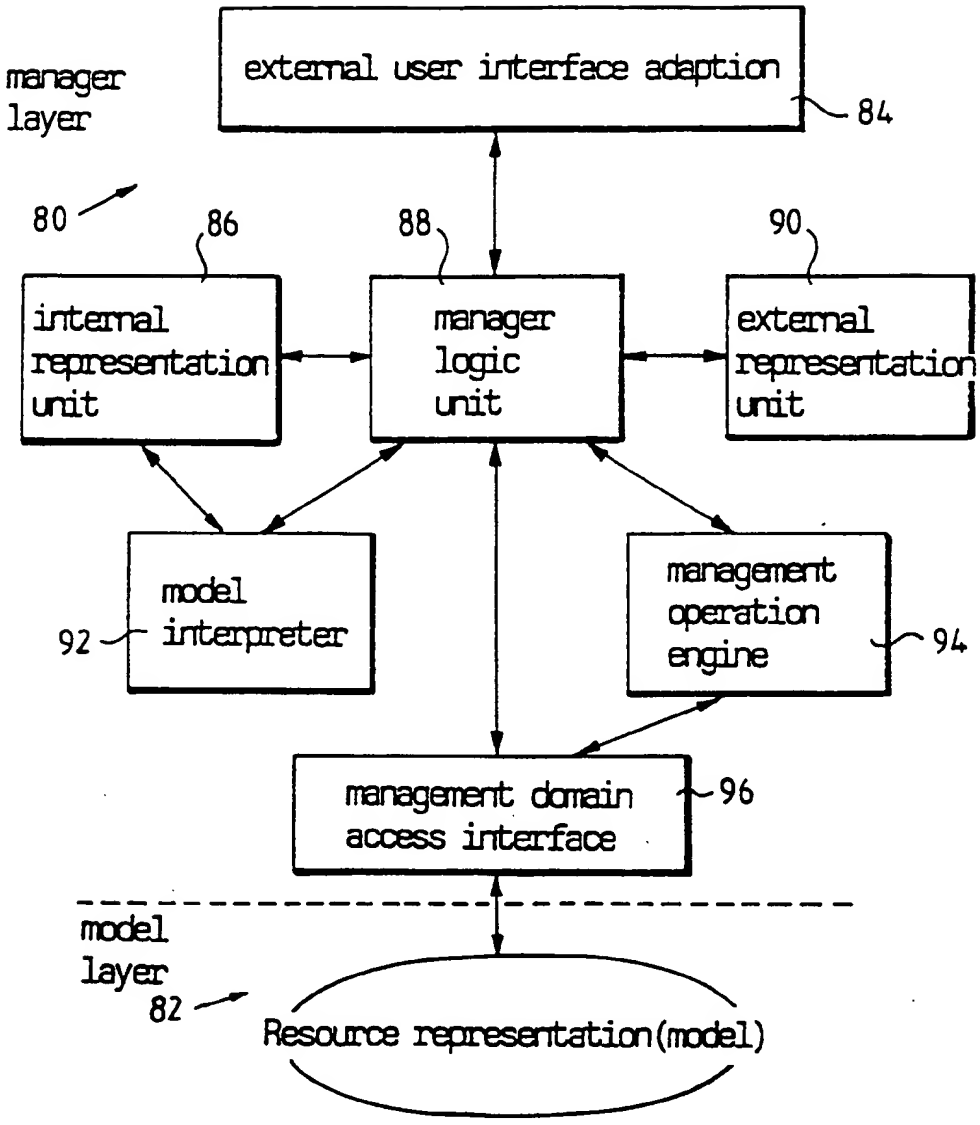

manager layer
external user interface adaption
84
80
86
88
90
internal representation unit
manager logic unit
external representation unit
model interpreter
92
management operation engine
94
management domain access interface
96
model layer
82
Resource representation (model)

Fig. 16

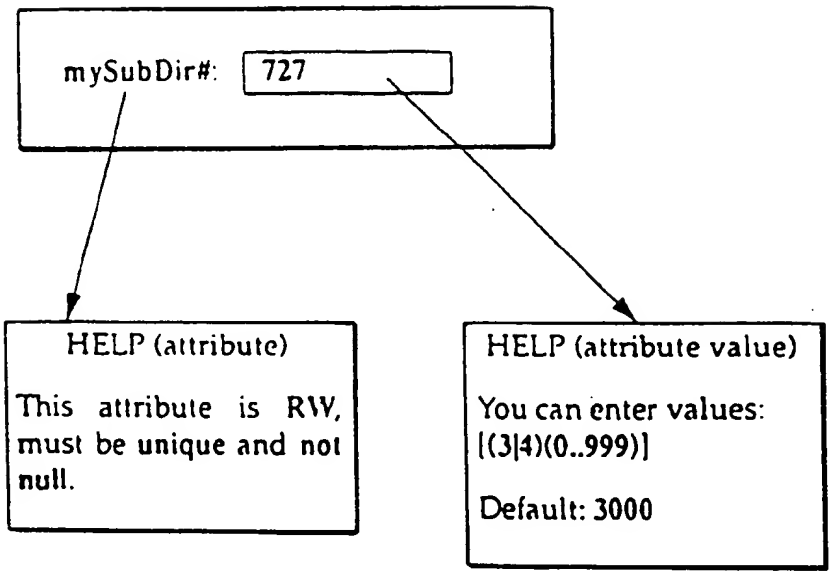

mySubDir#: 727
HELP (attribute)
This attribute is RW, must be unique and not null.
HELP (attribute value)
You can enter values: [(3|4)(0..999)]
Default: 3000

Fig. 17

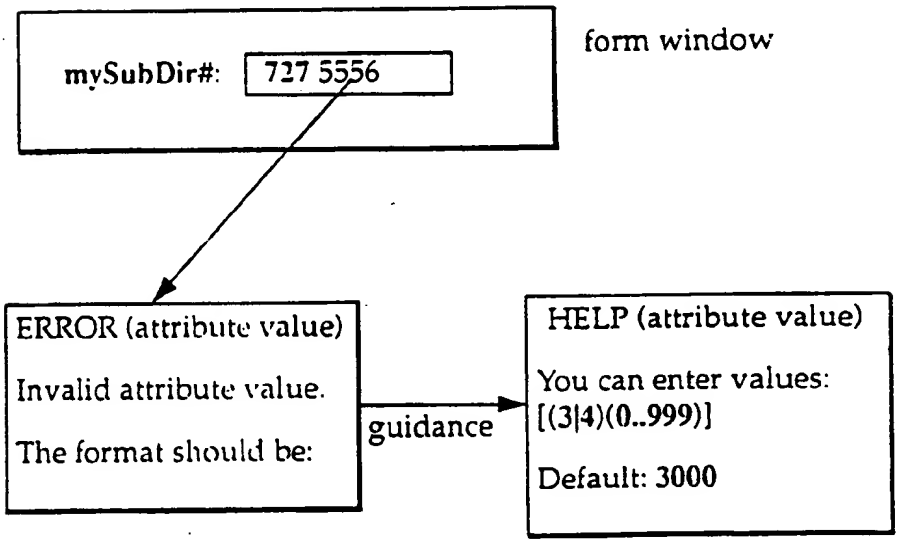

mySubDir#: 727 5556
form window
ERROR (attribute value)
Invalid attribute value.
The format should be:
guidance
HELP (attribute value)
You can enter values: [(3|4)(0..999)]
Default: 3000

Fig 18

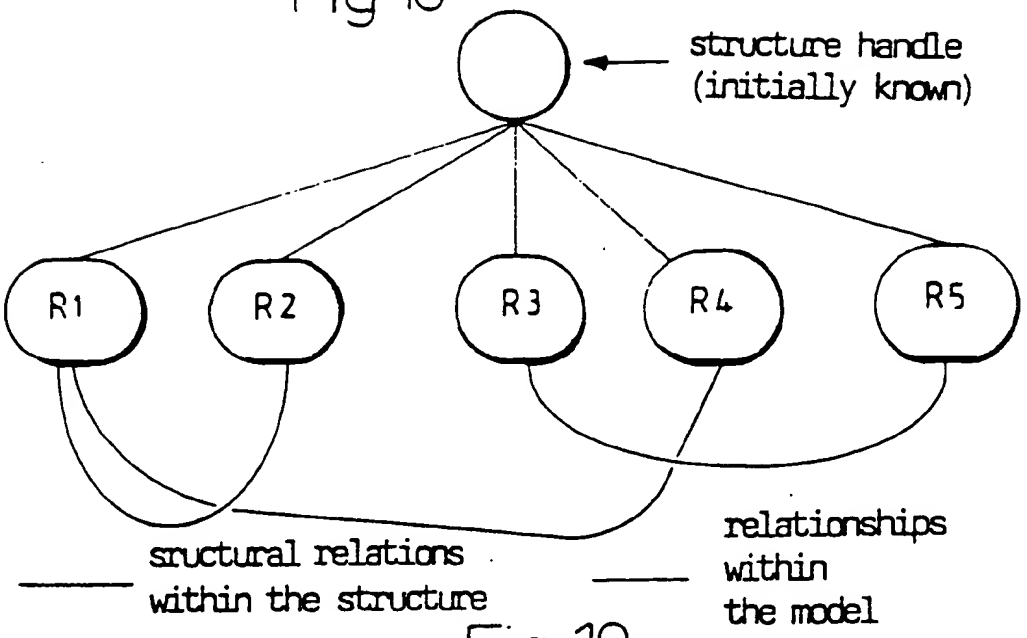

structure handle
(initially known)
R1
R2
R3
R4
R5
sructural relations
within the structure
relationships
within
the model

Fig. 19

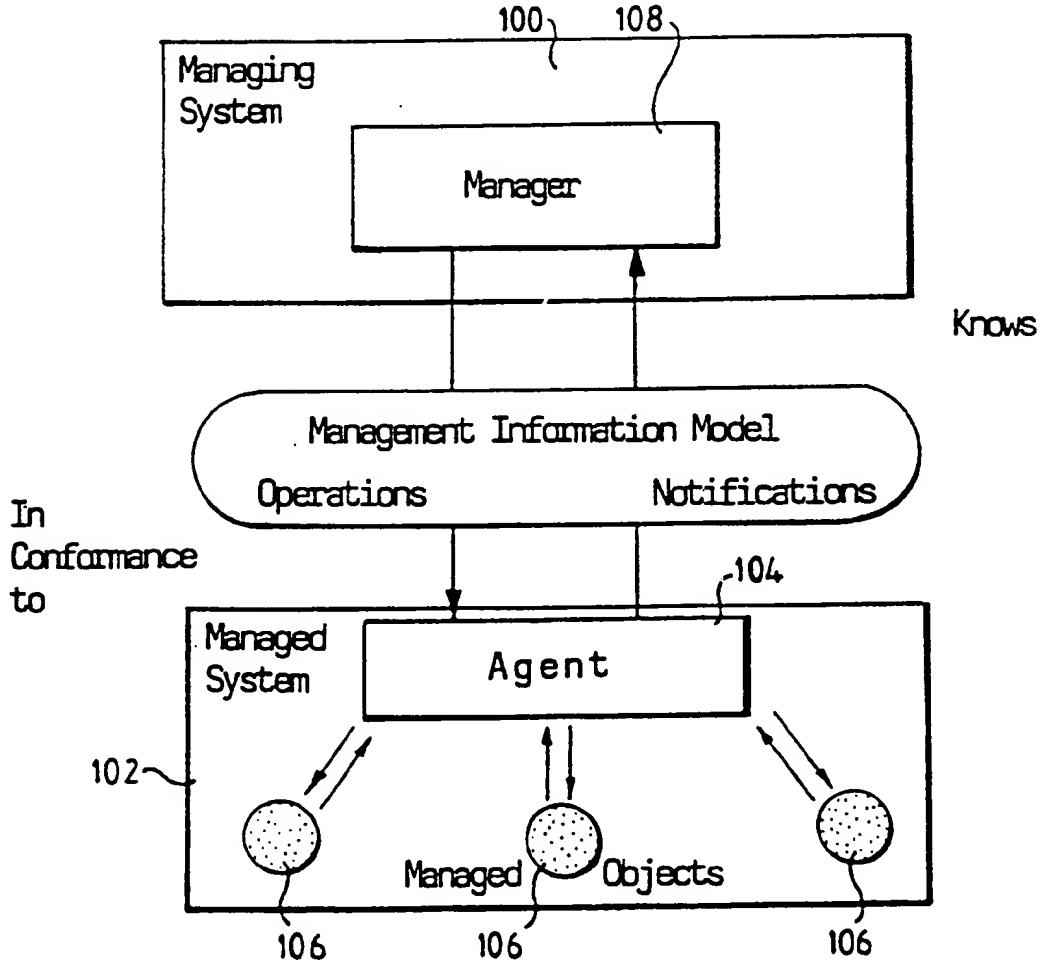

100
108
Managing
System
Manager
Knows
Management Information Model
Operations
Notifications
In
Conformance
to
104
Managed
System
Agent
102
Managed
Objects
106
106
106

Fig. 20

Object type A

Attributes
NoOfLinks

Relations
Object B

Actions
OpenLink
CloseLink

Object type B

Attributes
NoOfLinks
LinkID
Relations
Object A

Actions
SendData

Fig. 21

Attributes
ListOfAttributes
ListOfActions
ListOfRelations
etc.

Object type MIM Server

Fig. 22

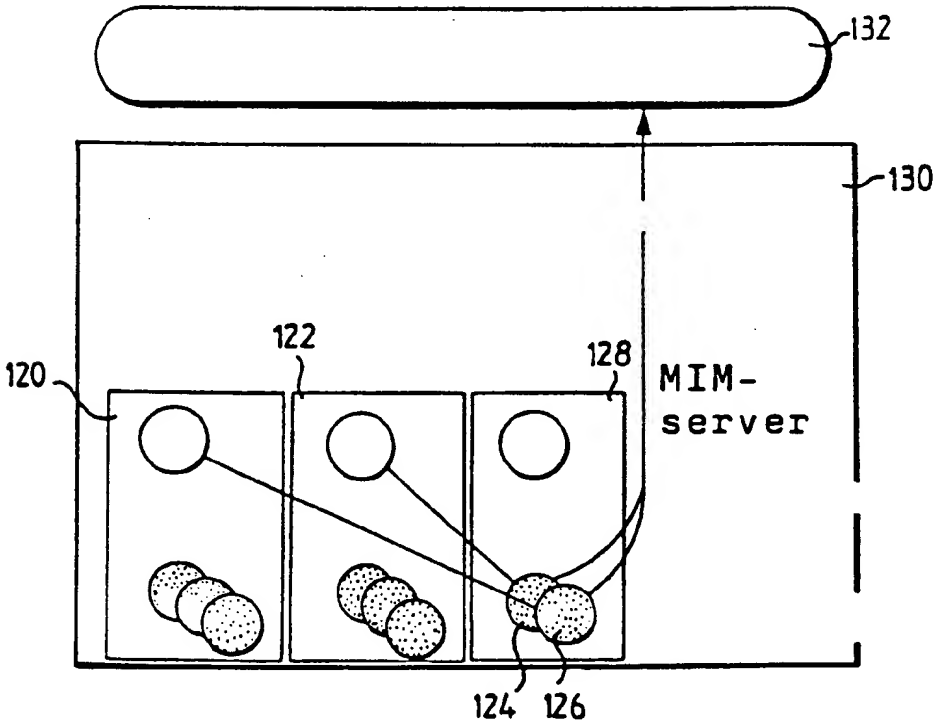


Fig. 23

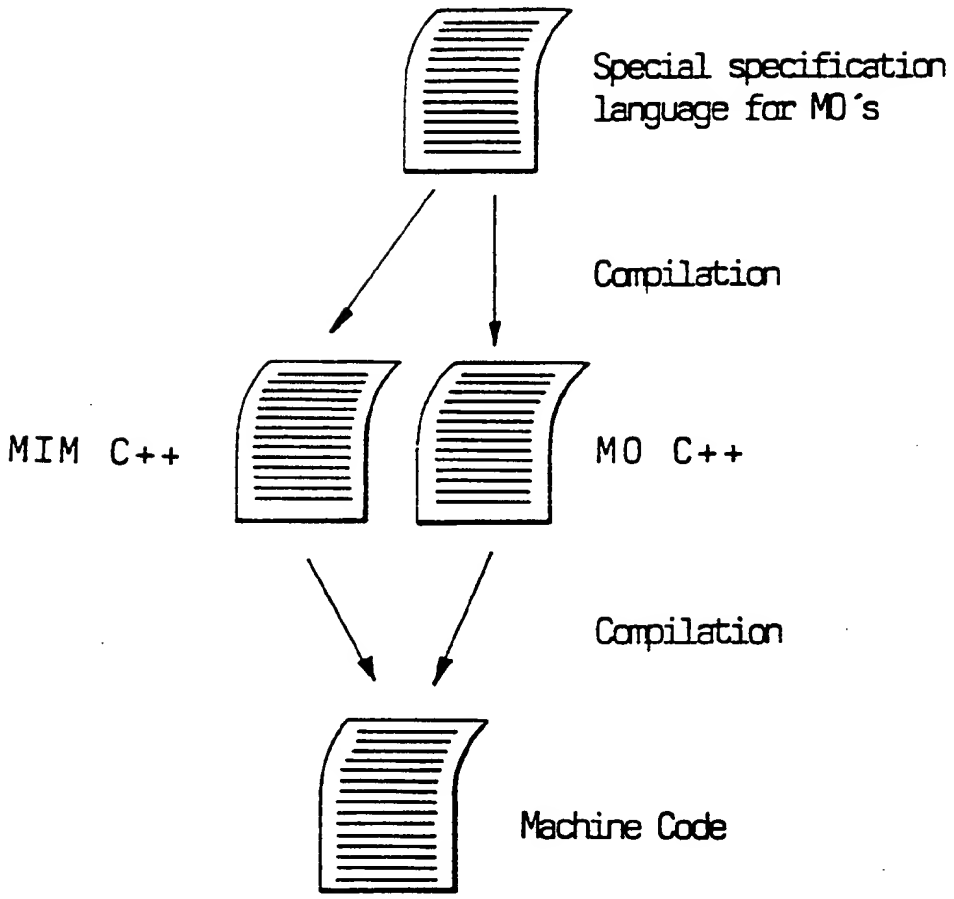

Special specification language for MO´s
Compilation
MIM C++
MO C++
Compilation
Machine Code

Fig. 24

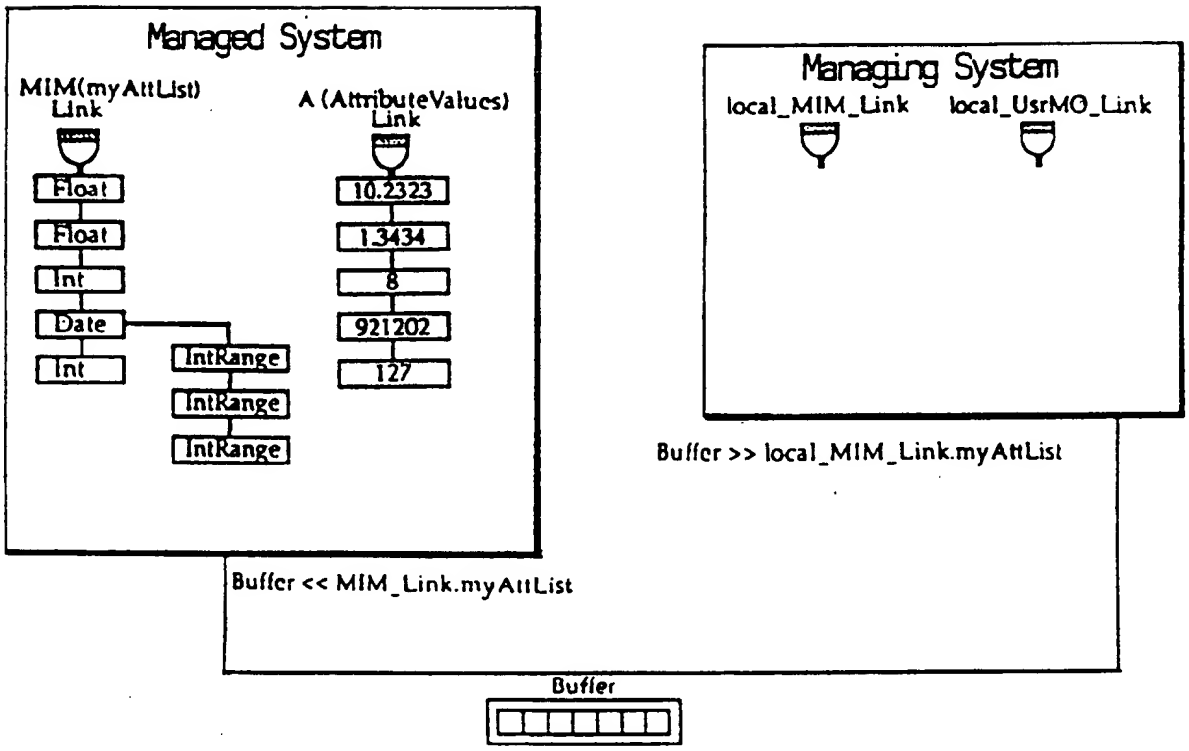

Managed System
MIM(myAttList) Link
A (AttributeValues) Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
10.2323
1.3434
8
921202
127
Managing System
local_MIM_Link
local_UsrMO_Link
Buffer >> local_MIM_Link.myAttList
Buffer << MIM_Link.myAttList
Buffer

Fig. 25

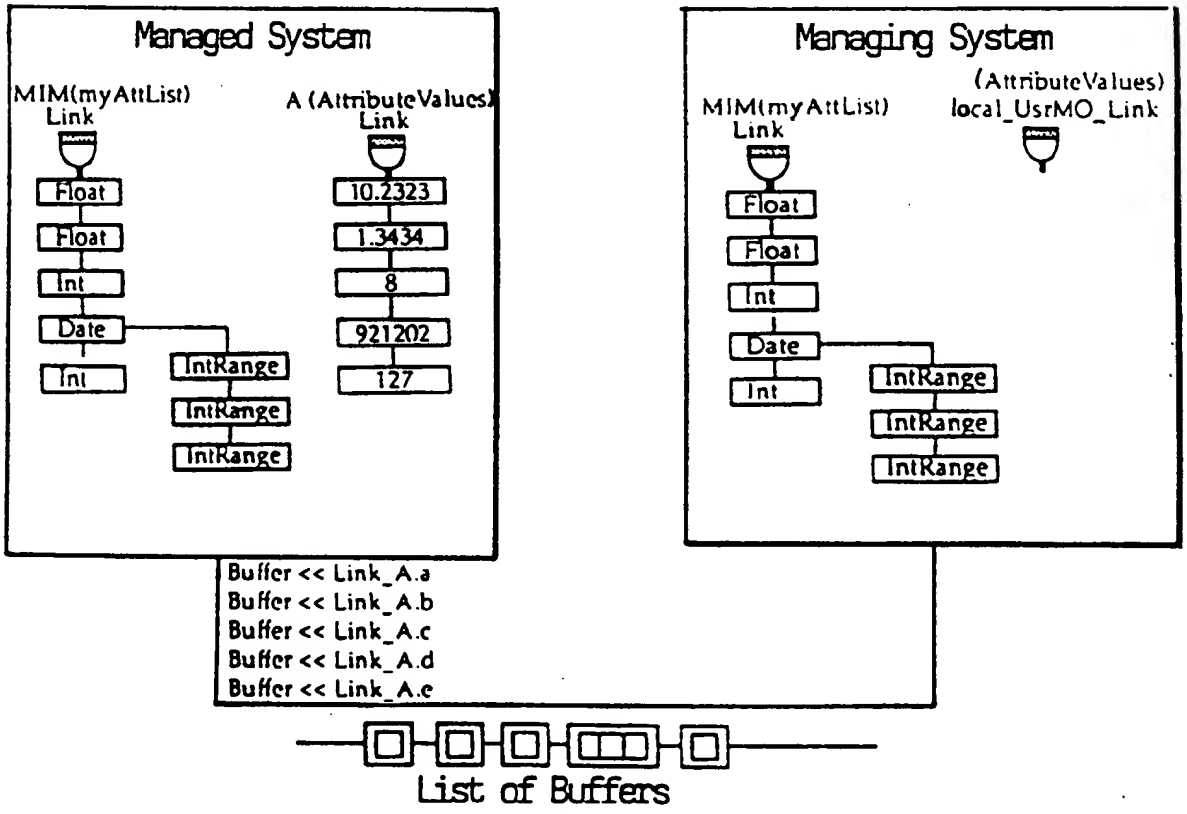

Managed System
MIM(myAttList) Link
A (AttributeValues) Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
10.2323
1.3434
8
921202
127
Managing System
(AttributeValues)
MIM(myAttList) Link
local_UsrMO_Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
Buffer << Link_A.a
Buffer << Link_A.b
Buffer << Link_A.c
Buffer << Link_A.d
Buffer << Link_A.e
List of Buffers

Fig. 26

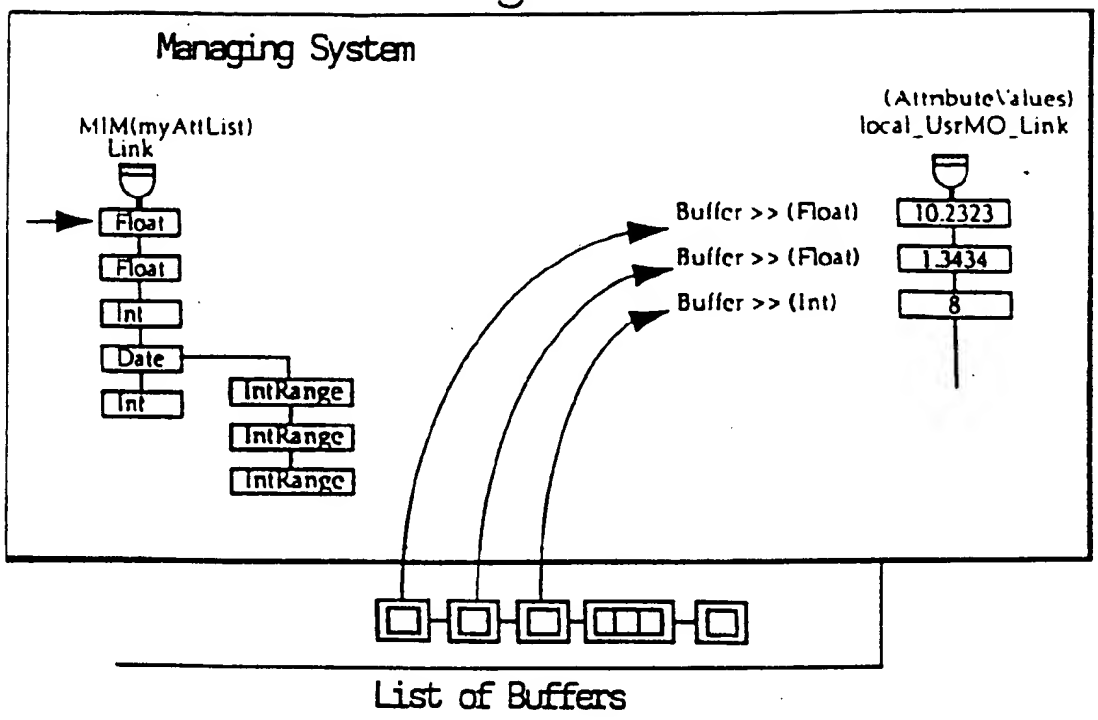

Managing System
MIM(myAttList)
Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
(AttributeValues)
local_UsrMO_Link
Buffer >> (Float)
Buffer >> (Float)
Buffer >> (Int)
10.2323
1.3434
8
List of Buffers

Fig. 27

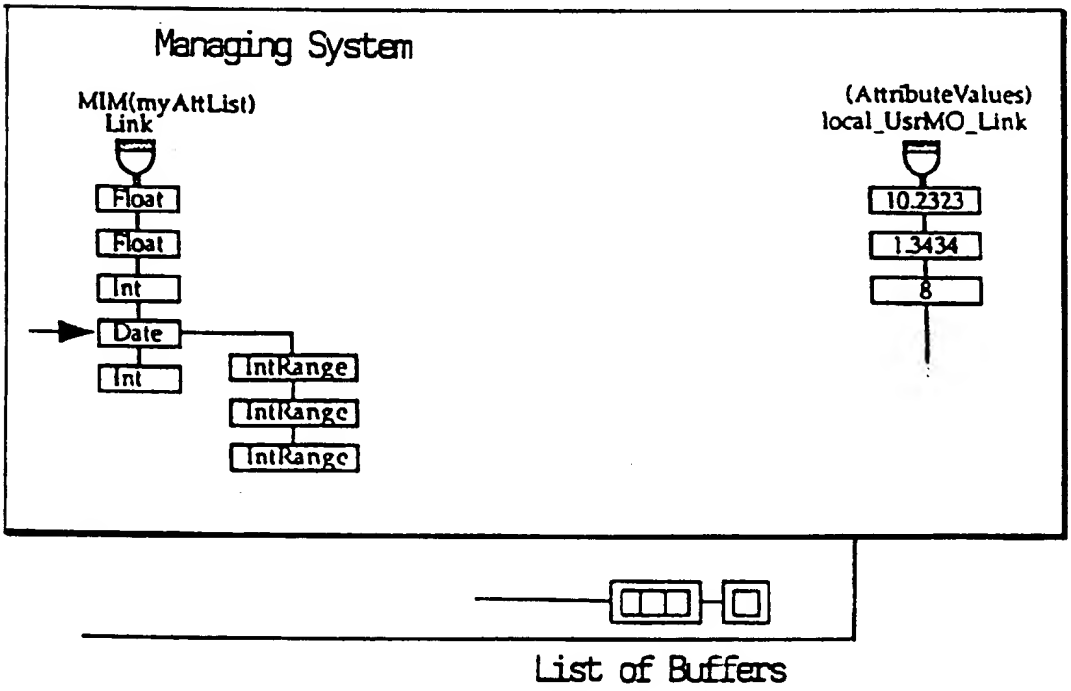

Managing System
MIM(myAttList)
Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
(AttributeValues)
local_UsrMO_Link
10.2323
1.3434
8
List of Buffers

Fig. 28

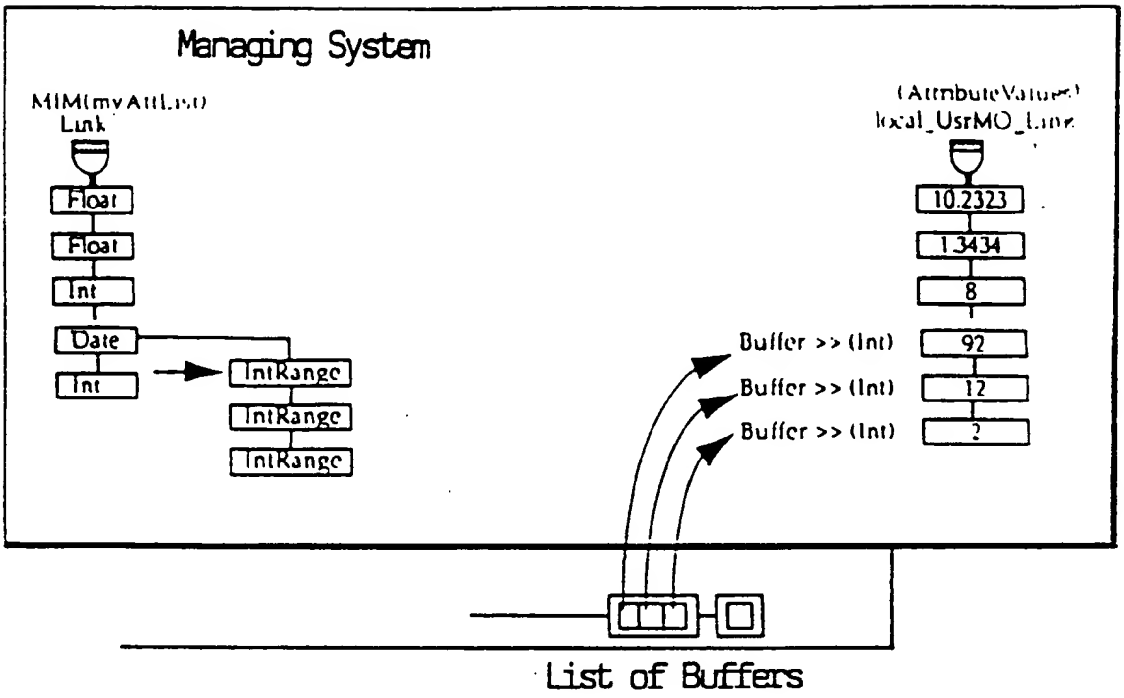

Managing System
MIM(myAttList)
Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
(AttributeValues)
local_UsrMO_Link
10.2323
1.3434
8
Buffer >> (Int)
92
Buffer >> (Int)
12
Buffer >> (Int)
?
List of Buffers

Fig. 29

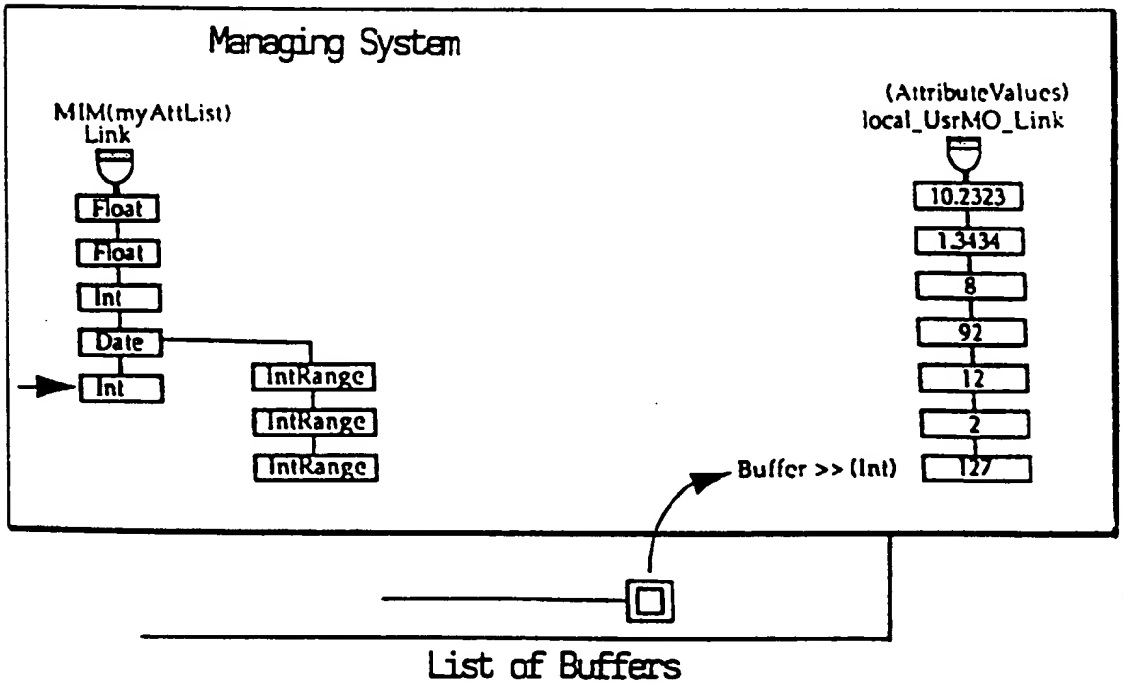

Managing System
MIM(myAttList)
Link
Float
Float
Int
Date
Int
IntRange
IntRange
IntRange
(AttributeValues)
local_UsrMO_Link
10.2323
1.3434
8
92
12
2
Buffer >> (Int)
127
List of Buffers

Fig. 30

```
1  OBJECT TYPE Subscriber IS
2
3  ATTRIBUTES
4    Number: NumberType;
5    AdmState: admStateType;
6    OpState: opStateType;
7    UsageState: usageStateType;
8    Line: REFERENCE LineDriver;
9  METHODS
10   LockRequest ();
11   Seise () RETURN Boolean;
12   Release ();
13
14 PERSISTENT PROPERTIES
15   PRIMARY KEY Number;
16   IMPLICIT UsageState;
17
18 MANAGED OBJECT PROPERTIES
19   READ-WRITE Line;
20   READ Number, AdmState, OpState,
     UsageState;
21   ACTIONS LockRequest;
22 END;
```

Fig.31

```
23    TYPE opStateType IS
24      ENUM disabled := 0, enabled := 1
25    END;
26
27    TYPE usageStateType IS
28      ENUM idle := 0, active := 1, busy := 2
29    END;
30
31    TYPE admStateType IS
32      ENUM locked := 0, unlocked := 1,
33        shuttingDown := 2
34    END;
35
36    TYPE NumberType IS
37        ARRAY OF BCDCodedDigits
38    END;
39
40    TYPE BCDCodedDigits IS
41        NATURAL RANGE 0..9
42    END;
```

Fig. 32

```
43    OBJECT TYPE Subscriber IS
44
45    ATTRIBUTES
46      Number: NumberType;
47      AdmState: admStateType;
48      OpState: opStateType;
49      UsageState: usageStateType;
50      Line: REFERENCE LineDriver;
51    METHODS
52      LockRequest ();
53      Seise () RETURN Boolean;
54      Release ();
55
56    PERSISTENT PROPERTIES
57      PRIMARY KEY Number;
58      IMPLICIT UsageState;
59
60    MANAGED OBJECT PROPERTIES
61*     READ-WRITE Line;
62*     READ Number, AdmState, OpState,
63*       UsageState;
64*     ACTIONS LockRequest;
65      PROHIBIT CREATE, DELETE;
66    END;
```

Fig. 33

```
67   OBJECT TYPE Subscriber IS
68
69   ATTRIBUTES
70     Number: NumberType;
71     AdmState: admStateType;
72     OpState: opStateType;
73     UsageState: usageStateType;
74     Line: REFERENCE LineDriver;
75   METHODS
76     LockRequest ();
77     Seise() RETURN Boolean;
78     Release();
79   PRE-CONDITIONS
80       SET Line TO NULL ONLY IF AdmState = locked;
81   PERSISTENT PROPERTIES
82     PRIMARY KEY Number;
83     IMPLICIT UsageState;
84
85   MANAGED OBJECT PROPERTIES
86     READ-WRITE Line;
87     READ Number, AdmState, OpState,
88       UsageState;
89     ACTIONS LockRequest;
90     PROHIBIT CREATE, DELETE;
91   END;
```

Fig. 34

```
92   OBJECT TYPE Subscriber IS
93
94   ATTRIBUTES
95     Number: NumberType;
96     AdmState: admStateType;
97     OpState: opStateType;
98     UsageState: usageStateType;
99     Line: REFERENCE LineDriver INVERSE OF Subsc;
100  METHODS
101    LockRequest ();
102    Seise () RETURN Boolean;
103    Release ();
104  PRE-CONDITIONS
105    SET Line TO NULL ONLY IF AdmState=locked;
106  POST-CONDITIONS
107    NOT (Line = NULL AND AdmState = unlocked);
108  PERSISTENT PROPERTIES
109    PRIMARY KEY Number;
110    IMPLICIT UsageState;
111
112  MANAGED OBJECT PROPERTIES
113    READ-WRITE Line;
114    READ Number, AdmState, OpState,
115      UsageState;
116    ACTIONS LockRequest;
117    PROHIBIT CREATE,DELETE;
118  PARTY TO LineAndSubscriber;
119  END;
```

Fig. 35

```
120
121    PRE-CONDITIONS
122      SET Line TO NULL ONLY IF AdmState = locked;
123    POST-CONDITIONS
124      NOT Line = NULL AND AdmState = unlocked;
125        RULE CHECK CONSISTENCY;
126    PERSISTENT PROPERTIES
127
128    END;
```

Fig. 36

```
129
130  PRE-CONDITIONS
131    SET Line TO NULL ONLY IF AdmState = locked;
132  POST-CONDITIONS
133    NOT Line = NULL AND AdmState = unlocked;
134      RULE MAINTAIN CONSISTENCY;
135  PERSISTENT PROPERTIES
136
137  END;
```

Fig. 37

```
138  OBJECT TYPE LineDriver IS
139
140  ATTRIBUTES
141    Cicuit: CircuitType;
142    AdmState: admStateType;
143    OpState: opStateType;
144    UsageState: usageStateType;
145    Subsc: REFERENCE Subscriber INVERSE OF Line;
146  METHODS
147    LockRequest();
148    Seise() RETURN Boolean;
149    Release();
150  PRE-CONDITIONS
151    SET Subsc TO NULL ONLY IF AdmState = locked;
152  POST-CONDITIONS
153    NOT (Subsc = NULL AND AdmState = unlocked);
154  PERSISTENT PROPERTIES
155    PRIMARY KEY Circuit;
156    IMPLICIT UsageState;
157
158  MANAGED OBJECT PROPERTIES
159    READ-WRITE Subsc;
160    READ Number, AdmState, OpState,
161      UsageState;
162    ACTIONS LockRequest;
163  PARTY TO LineAndSubscriber;
164  END;
```

Fig. 38

```
165  DEPENDENCY SCHEMA LineAndSubscriber IS
166    FOR ALL LineDriver (1), Subscriber (s);
167    RELATIONS 1.Subsc = s;
168
169    POST-CONDITIONS
170      NOT (1.AdmState = unlocked AND
171       s.AdmState = locked);
172      RULE MAINTAIN CONSISTENCY;
173  END;
```

Fig. 39

```
174
175  POST-CONDITIONS
176    WHEN LockRequest
177      AdmState = locked OR
178      AdmState = shuttingDown;
179
```

Fig. 40

```
180
181  POST-CONDITIONS
182    WHEN CREATE
183      Line/= NULL;
184
```

Fig. 41

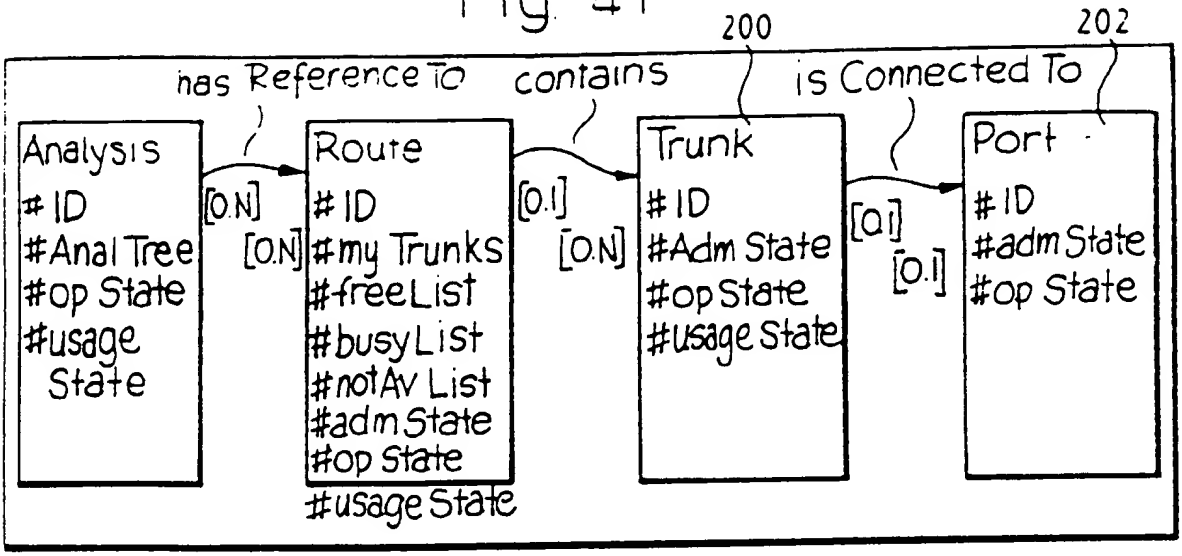

200
202
has Reference To
contains
is Connected To
Analysis
#ID
#Anal Tree
#op State
#usage State
[0.N]
[0.N]
Route
#ID
#my Trunks
#free List
#busy List
#not Av List
#adm State
#op State
#usage State
[0.1]
[0.N]
Trunk
#ID
#Adm State
#op State
#usage State
[0.1]
[0.1]
Port
#ID
#adm State
#op State

Fig. 42

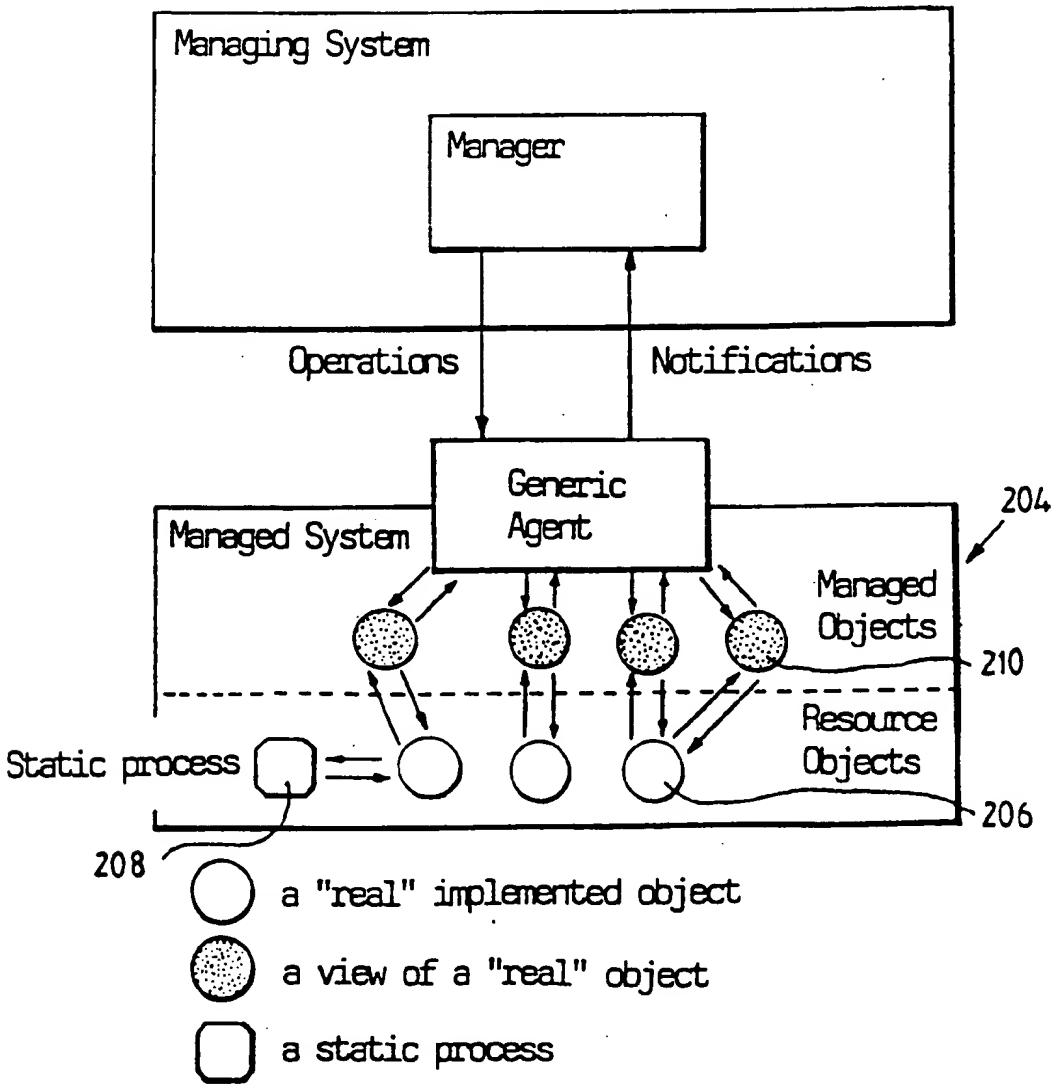

Managing System
Manager
Operations
Notifications
Generic Agent
Managed System
204
Managed Objects
210
Resource Objects
Static process
206
208
a "real" implemented object
a view of a "real" object
a static process

Fig. 43

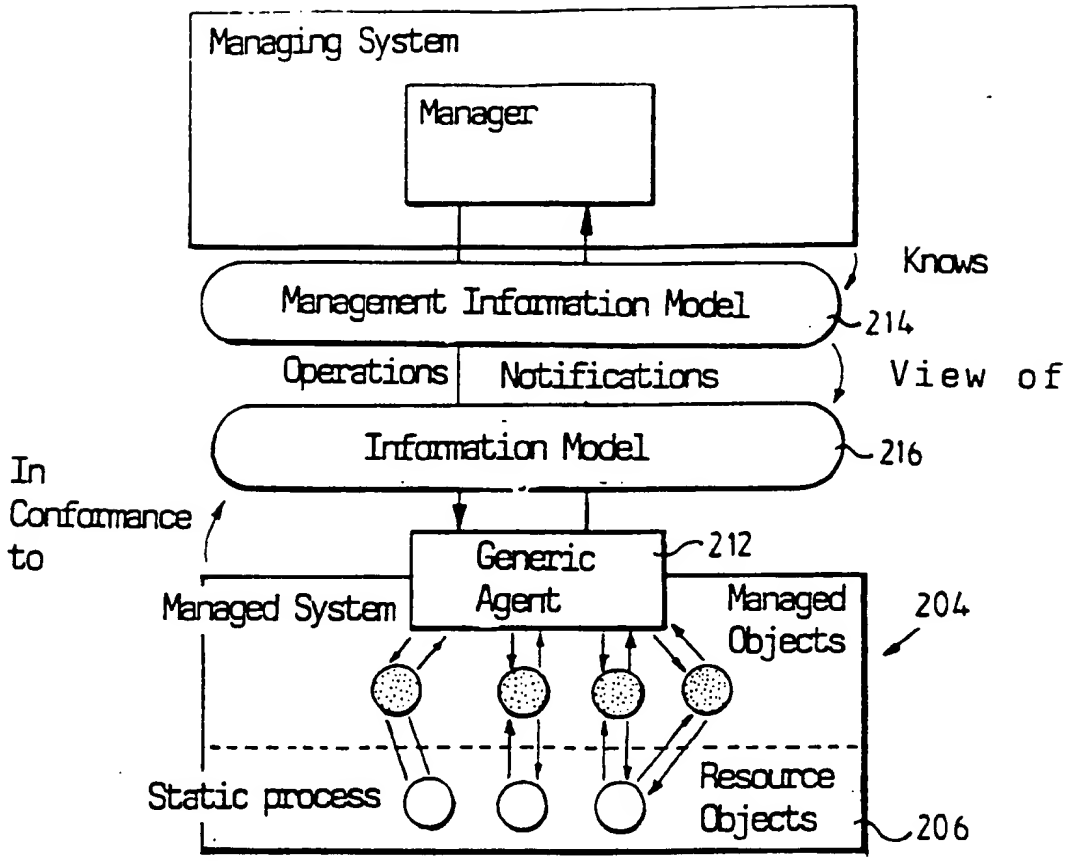

Managing System
Manager
Knows
Management Information Model
214
Operations
Notifications
View of
Information Model
216
In
Conformance
to
Generic
Agent
212
Managed System
Managed
Objects
204
Static process
Resource
Objects
206

Fig. 44

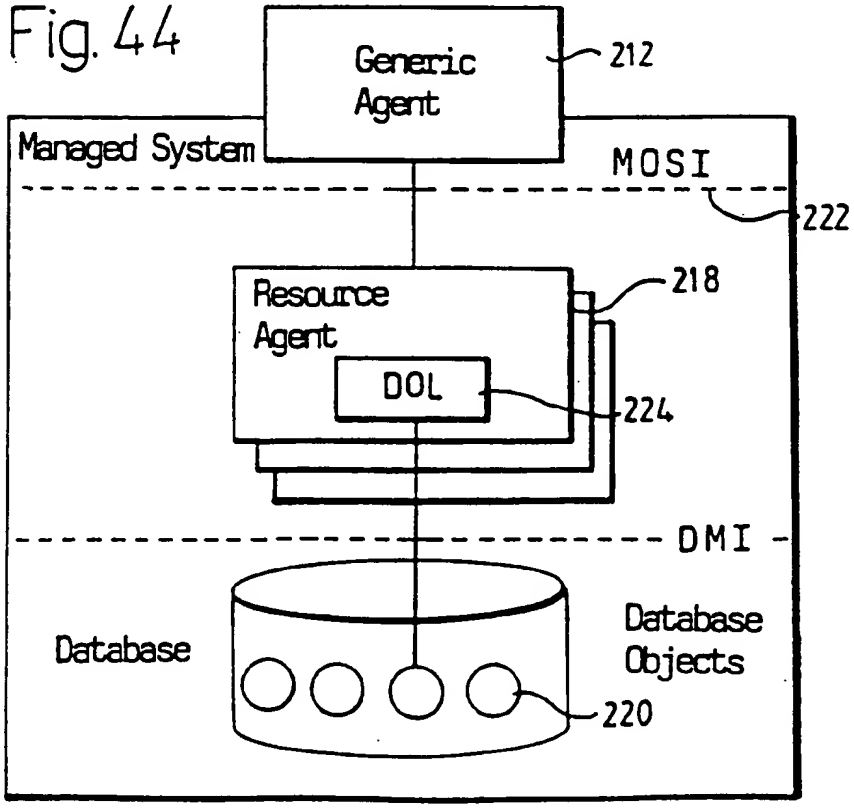

Generic
Agent
212
Managed System
MOSI
222
Resource
Agent
218
DOL
224
DMI
Database
Database
Objects
220

Fig. 45

```
1  OBJECT TYPE AnObject IS
2    BASE BaseObject;
3    ATTRIBUTES
4      attr1 : Atype;
5      attr2 : Integer;
6      attr3 : Integer;
7
8    METHODS
9      m1 (IN anArg : ArgType);
10     m2 (IN anArg : AnotherArgType)
       RETURNS AreturnType;
11
12   PRE-CONDITIONS
13     m1 (aValue) ONLY IF
       attr2 = attr3;
14
15   POST-CONDITIONS
16     attr2 >= attr3;
17
18 END;
```

Fig. 46

```
20 OBJECT TYPE ResourceA IS
21    ATTRIBUTES
22      key : KeyType;
23      admState : AdministrativeState;
24      opState : OperationalState
25      Bref : REFERENCE TO ResourceB
                INVERSE OF Aref;
26    PRIMARY KEY key;
27
28    METHODS
29      allocate() RETURNS Boolean;
30
31    PRE-CONDITIONS
32      deleteObject () ONLY IF
        admState = locked;
33
34    POST-CONDITIONS
35      NOT (admState=unlocked
        AND Bref=NULL);
36
37    PARTY TO ResourceAandB;
38
39 END;
```

Fig. 47

```
40 OBJECT TYPE ResourceB IS
41   ATTRIBUTES
42     key : KeyType;
43     admState : AdministrativeState;
44     opState : OperationalState:
45     Aref : REFERENCE TO ResourceA
                INVERSE OF Bref;
46   PRIMARY KEY key;
47
48   METHODS
49     allocate() RETURNS Boolean;
50
51   PRE-CONDITIONS
52     deleteObject() ONLY IF
       admState=locked;
53
54   PARTY TO ResourceAandB;
55
56 END;
```

Fig. 48

```
57 DEPENDENCY SCHEMA ResourceAandB IS
58   FOR ALL ResourceA(a), ResourceB(b);
59   RELATIONS a.Bref = b;
60
61   POST-CONDITIONS
62     NOT (a.admState = unlocked AND
63          b.admState = locked);
64
65     NOT (a.opState = enabled AND
66            b.opState = disabled);
67
68 END;
```

Fig.50

```
69 DEPENDENCY SCHEMA ResourceAandB IS
70   FOR ALL ResourceA(a), ResourceB(b);
71   RELATIONS a.Bref = b;
72
73   POST-CONDITIONS
74     NOT (a.admState=unlocked AND
75          b.admState=locked)
76     RULES
77      WHEN COMMIT THEN CHECK CONSISTENCY
78
79     NOT (a.opState=enabled AND
89          b.opState=disabled)
81     RULES
82      WHEN b.opState=disabled THEN CONCLUDE
        a.opState=disabled,
83      WHEN COMMIT AND a.opState=enabled
        THEN CHECK CONSISTENCY
84
85 END;
```

Fig.49

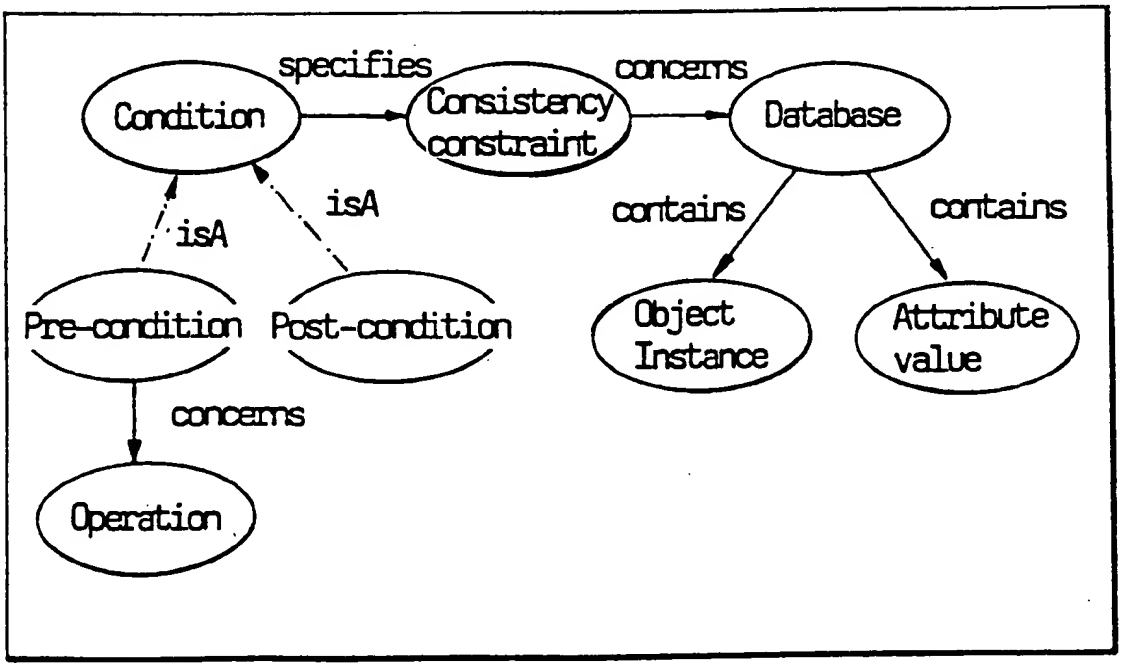


Fig. 51

```
86 OBJECT TYPE ResourceA IS
87   ATTRIBUTES
88     key : KeyType;
89     admState : AdministrativeState;
90     internalOpState : OperationalState;
91     resourceBstate : OperationalState;
92     DERIVED opState=IF (internalOpState=
93     disabled OR resourceBstate=disabled)
94                        THEN disabled
95                        ELSE enabled
96     Bref : REFERENCE TO ResourceB
              INVERSE OF Aref;
97
98   PRIMARY KEY key;
99
100  METHODS
101    allocate() RETURNS Boolean;
102
103  PRE-CONDITIONS
104   deleteObject() ONLY IF admState=locked;
105
106  POST-CONDITIONS
107    NOT (admState=unlocked AND Bref=NULL);
108
109  PARTY TO ResourceAandB;
110END;
```

Fig. 52

```
111DEPENDENCY SCHEMA ResourceAandB IS
112.....
113.....
114   NOT (a,opState = enabled AND
115         b.opState = disabled)
116   RULES
117    WHEN b.opState=NewValue
118    Then CONCLUDE a.resourceBstate=
       NewValue);
119
120END;
```

Fig.53

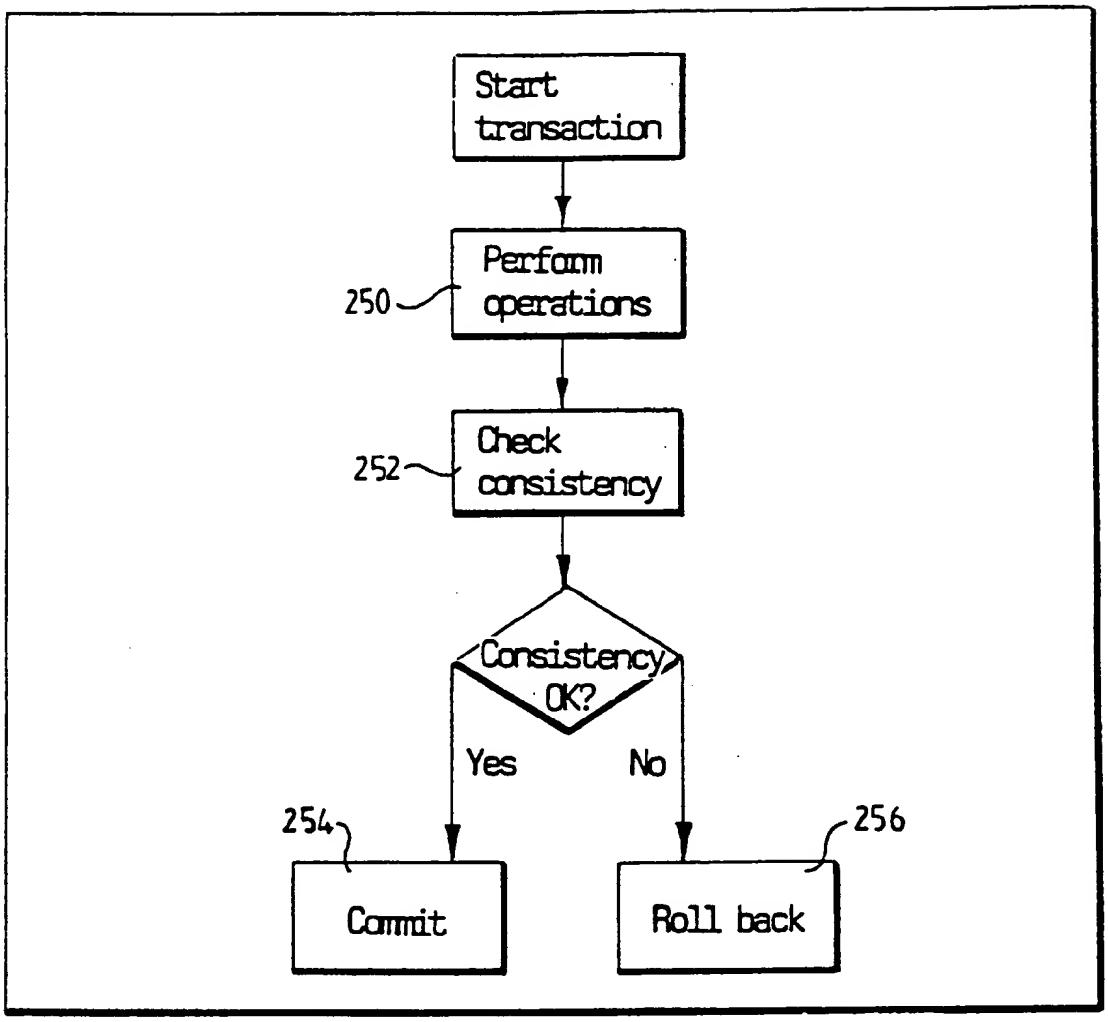

Start transaction
Perform operations
250
Check consistency
252
Consistency OK?
Yes
No
254
256
Commit
Roll back

Fig.54

```
121 ifndef _ResourceA_hh_
122 define _ResourceA_hh_
123
124 include <PredefinedTypes.hh>
125 include "MOstateTypes.hh"
126
127class ResourceB;
128
129// OBJECT TYPE ResourceA
130
131class ResourceA
132(
133public
134   static ResourceA* open(Mode,const
135   KeyType&, DbTransaction*transaction=NULL);
136   void deleteObject ();
137   virtual Boolean checkConsistency
      (ErrorMessage&);
138   Boolean allocate ();
139
140   KeyType get Key ();
141   AdministrativeState getAdmState();
142   void setAdmState (const
      AdministrtiveState);
143   OperationalState getOpState();
144   void setOpState(const
      OperationalState);
145   ResourceB* getBref();
146   void setBref(ResourceB*);
147
148private
149   .....
150   .....
151);
152
153 endif
```

Fig. 55

```
155 include "ResourceA.hh"
156
157//////////////////////////
158/////////
159//
160// ResourceA: METHOD checkConsistency
161//
162
163Boolean ResourceA: :checkConsistency
            (ErrorMessage& message)
164(
165   Boolean flag = true;
166
167   flag = !(getAdmState() == unlocked
      && getBreaf() == NULL;
168   if (!flag) (
169     createErrorMessage(1,message);
170   )
171   return flag;
172)
173
174
```

Fig. 56

```
175 ifndef _ResourceB_hh_
176 define _ResourceB_hh_
177
178 include <PredefinedTypes.hh>
179 include "MOstateTypes.hh"
180
181class ResourceA;
182
183// BOJECT TYPE ResourceB
184
185class ResourceB
186(
187public
188   static ResourceB* open(Mode,const KeyType&,
189   DbTransaction*transaction=NULL);
190   void deleteObject();
191   Boolean allocate();
192   virtual Boolean checkConsistency
      (ErrorMessage&);
193
194   KeyType getKey ();
195   AdministrativeState getAdmState();
196   void setAdmState(const
      AdministrativeState);
197   OperationalState getOpState ();
198   void setOpState (const OperationalState);
199   ResourceA* getAref();
200   void setAref(ResourceA*);
201
202private
203   void propagateOpState
      (const OperationalState);
204   void opState(const OperationalState);
205.....
206.....
207);
208
209 endif
```

Fig.57

```
211 include "ResourceB.hh"
212 include "ResourceA.hh"
213
214//////////////////////
215//
216// ResourceB: METHOD setOpState
217//
218
219void ResourceB: setOpState(const
                  OperationalState newValue)
220(
221   OperationalState oldValue=getOpState();
222
223   opState(newValue);
224   if (newValue==disabled && newValue
          != oldValue)
225   (
226     PropagateOpState (newValue);
227   )
228)
229
230//////////////////////
231//
232// ResourceB: METHOD propagateOpState
233//
234
235void ResourceB: .PropagateOpState
     (const OperationalState
236(
237   getAref () ->setOpState (newValue);
238)
239
```

Fig. 58

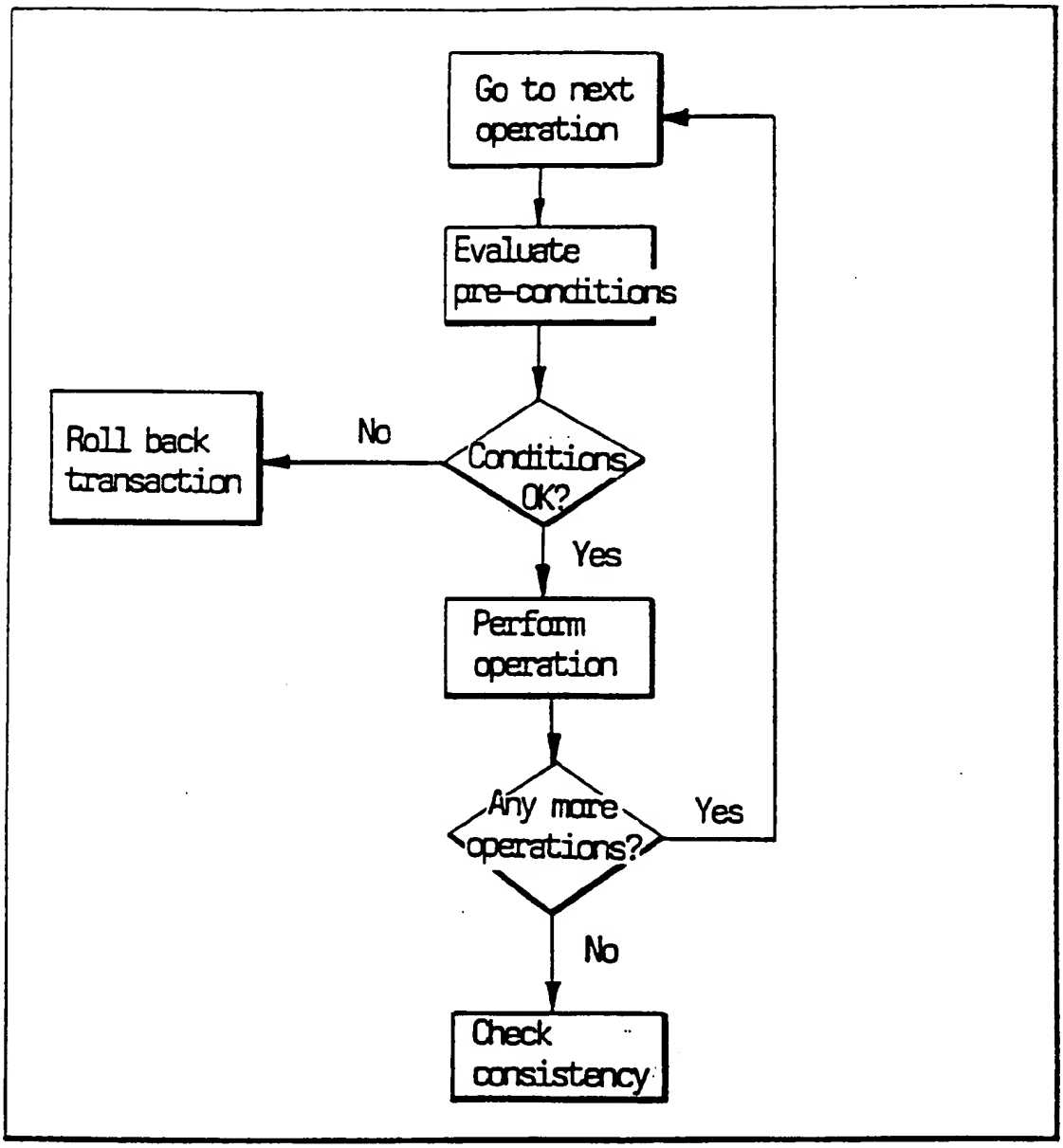

Go to next operation
Evaluate pre-conditions
Conditions OK?
No
Roll back transaction
Yes
Perform operation
Any more operations?
Yes
No
Check consistency

Fig. 59

```
240 ifndef _ResourceA_hh_
241 define _ResourceA_hh_
242
243 include <PredefinedTypes.hh>
244 include "MOstateTypes.hh"
245
246class ResourceB;
247
248// OBJECT TYPE ResourceA
249
250class ResourceA
251(
252public
253   static ResourceA* open(Mode,const
254   KeyType&, DbTransaction* transaction=NULL);
     =NULL);
255   void deleteObject ();
256   virtual Boolean checkConsistency
     (ErrorMessage&);
257   Boolean allocate ();
258
259   KeyType getKey();
260   AdministrativeState getAdmState();
261   void setAdmState(const
     AdministrativeState);
262   OperationalState getOpState)(;
263   void setOpState(const
     OperationalState),
264   ResourceB* getBref();
265   void setBref(ResourceB*);
266
267private
268   void deleteObjectCondition();
269   void reallyDeleteObject();
270);
271
272 endif
```

Fig. 60

```
274 include "ResourceA.hh"
275
276//////////////////////////
277//
278// ResourceA. METHOD remove
279//
280
281void ResourceA: :deleteObject()
282
283  deleteObjectcondition();
284  reakkyDeleteObject();
285)
286
287//////////////////////////
288//
289// ResourceA: METHOD
       deleteObjectCondition
290//
291
292void ResourceA: :deleteObjectCondition()
293(
294  AdministrativeState admStateValue=
     admStateDatabaseValue();
295
296  if (admStateValue != locked)
297  (
298    throw ErrorMessage(2,admStateValue);
299  )
300)
```

Fig.61

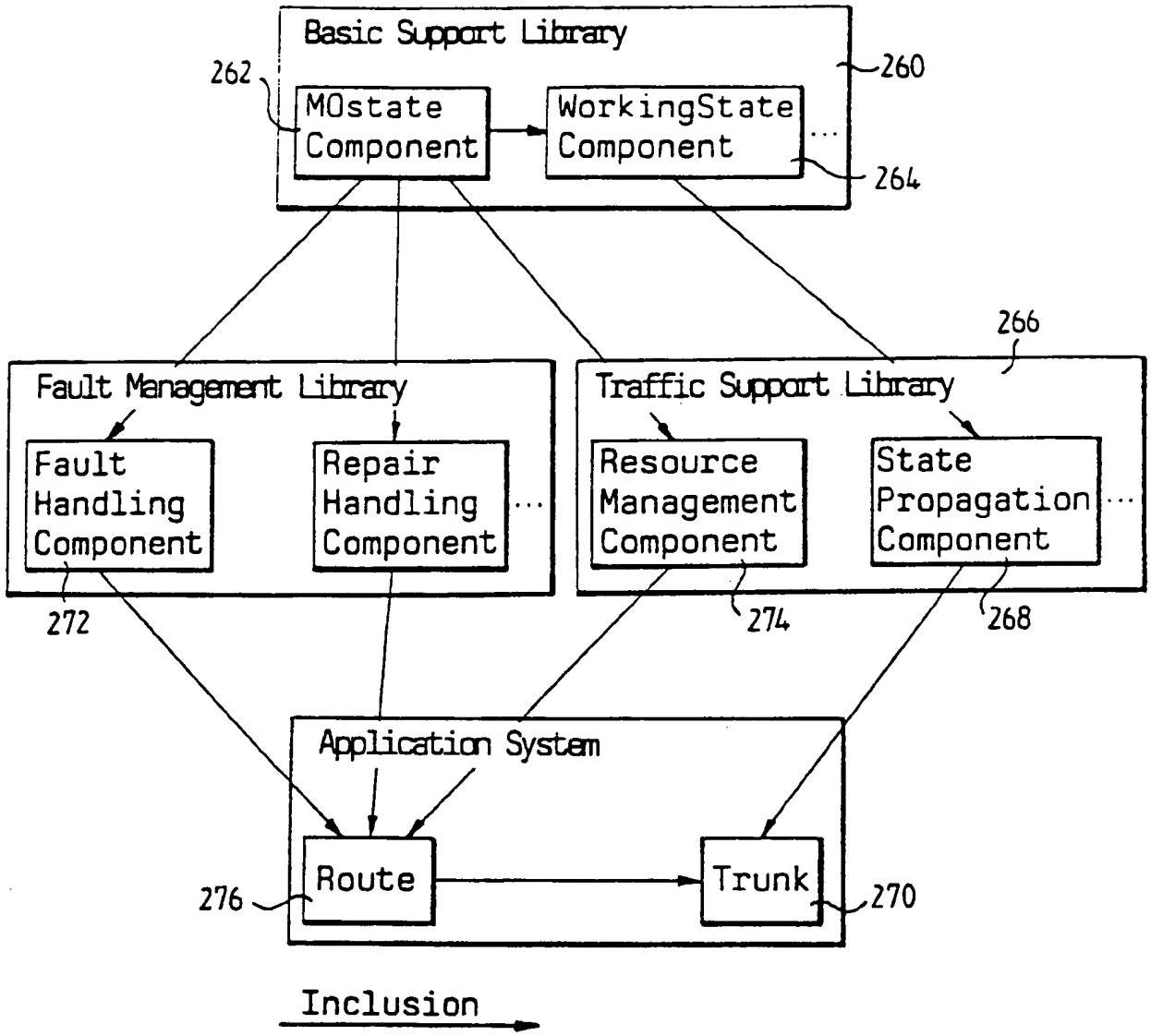

Basic Support Library
262
MOstate Component
WorkingState Component
260
264
266
Fault Management Library
Fault Handling Component
Repair Handling Component
Traffic Support Library
Resource Management Component
State Propagation Component
272
274
268
Application System
Route
Trunk
276
270
Inclusion

Fig.62

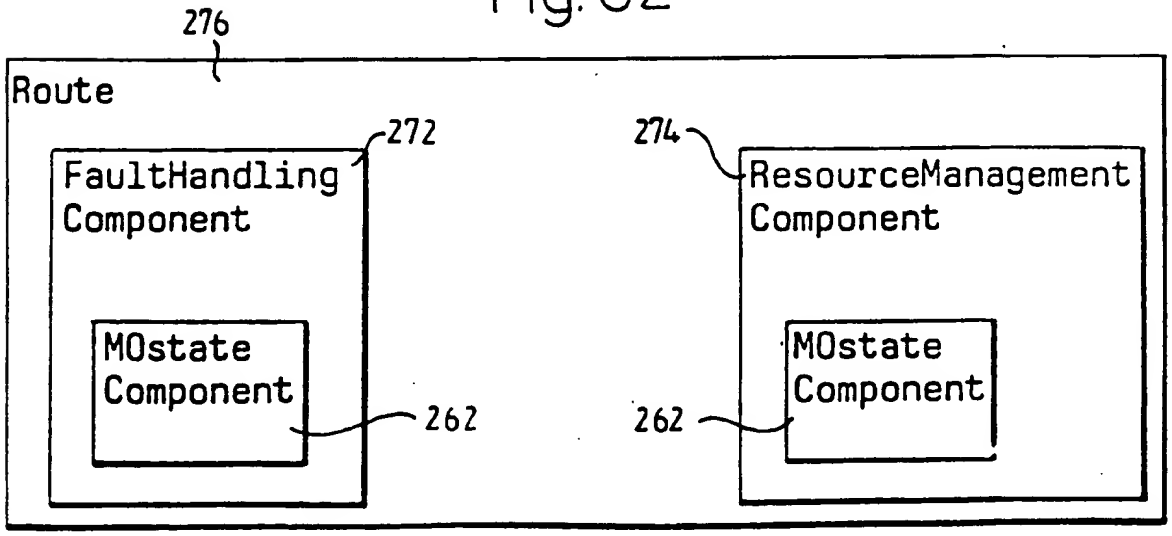

276
Route
272
FaultHandling Component
MOstate Component
262
274
ResourceManagement Component
MOstate Component
262

Fig. 63

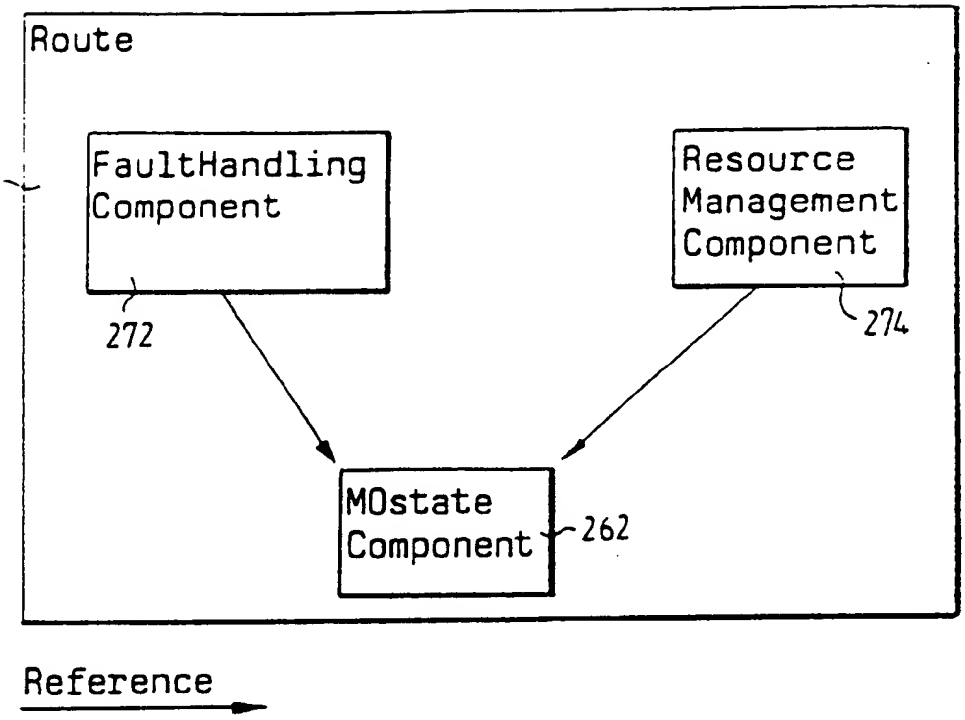


Fig. 64

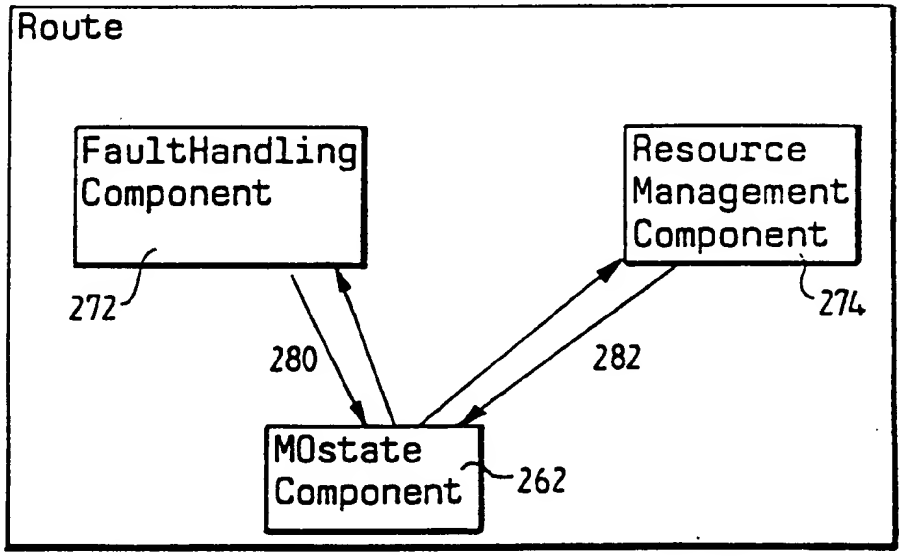


Fig. 65

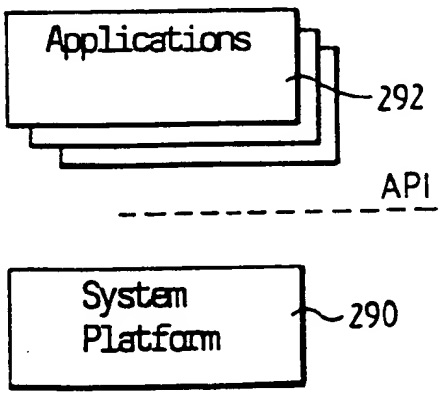


Fig.66

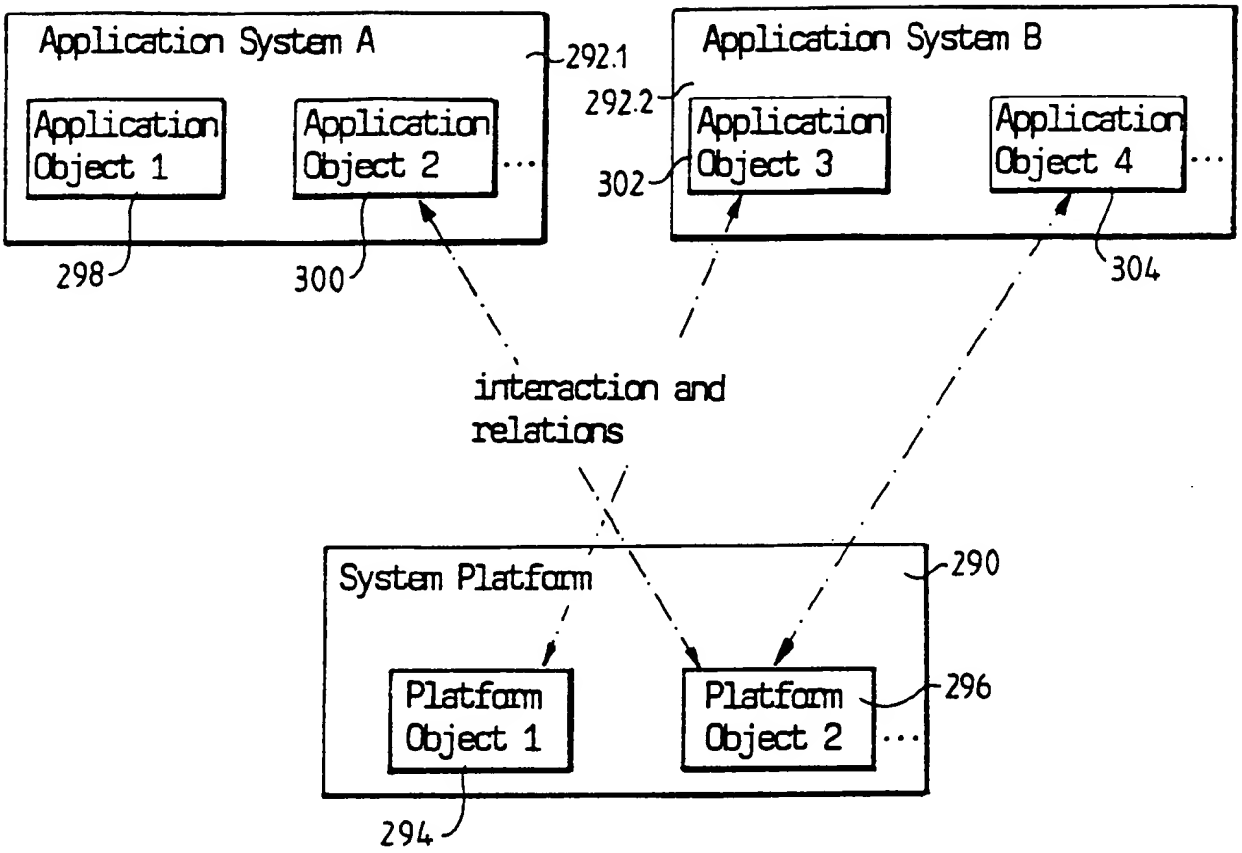

Application System A
292.1
Application Object 1
Application Object 2
298
300
Application System B
292.2
Application Object 3
Application Object 4
302
304
interaction and relations
System Platform
290
Platform Object 1
Platform Object 2
296
294

Fig.67

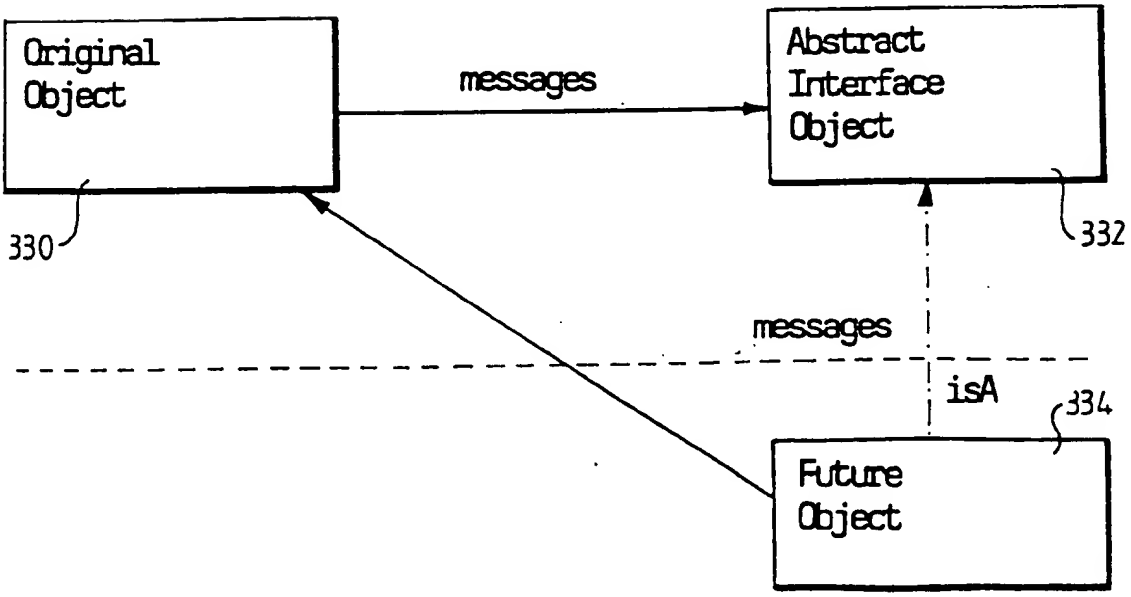

Original Object
messages
Abstract Interface Object
330
332
messages
isA
334
Future Object

Fig. 68

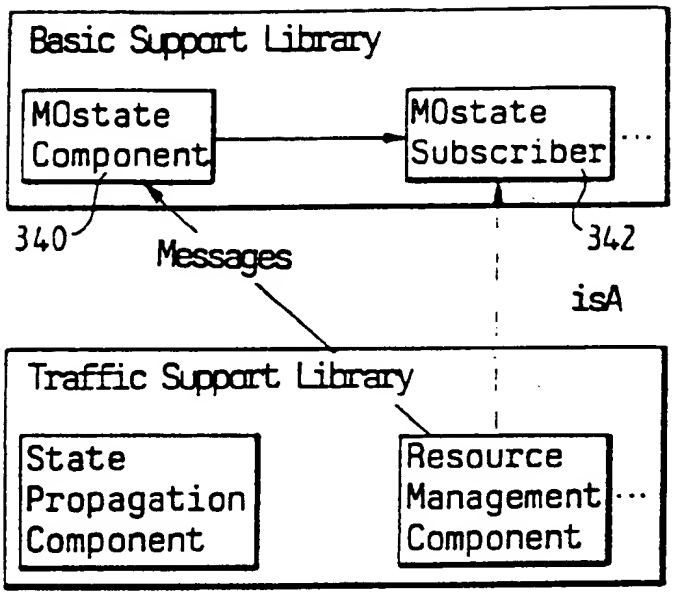


Fig. 69

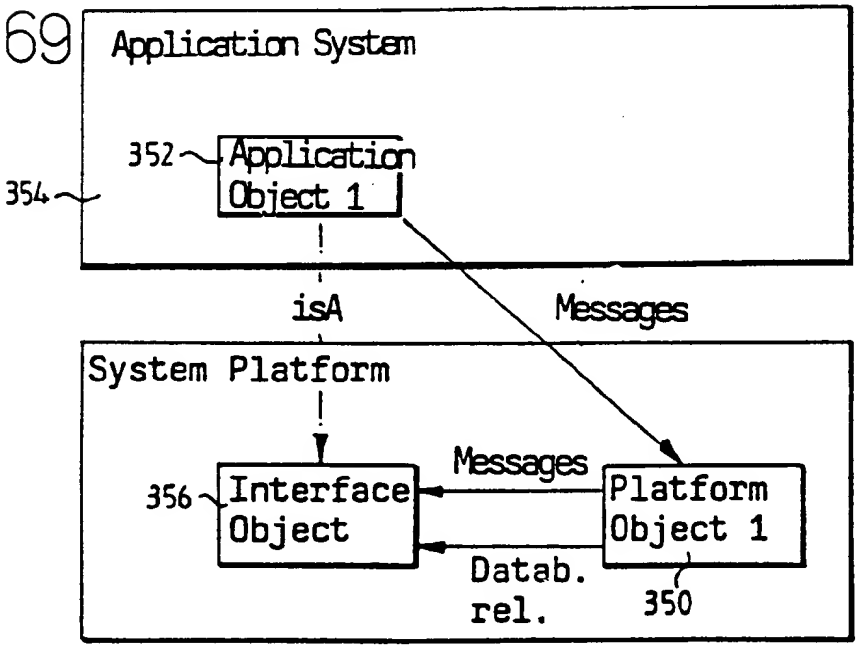


Fig. 70

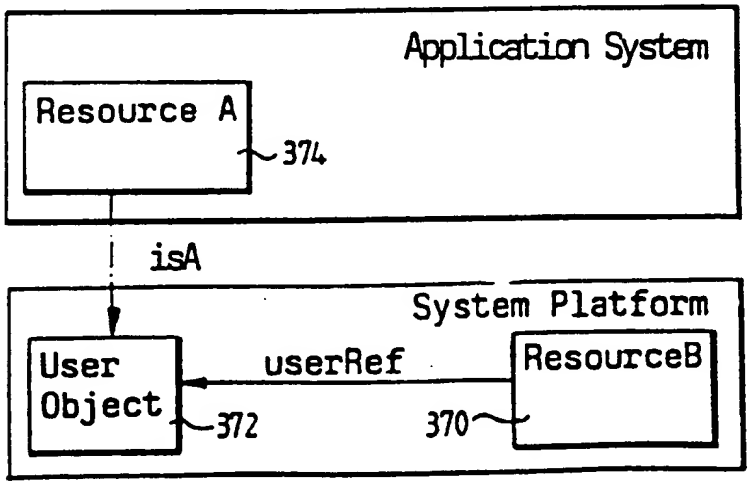


Fig.71

```
1   OBJECT TYPE ResourceB IS
2     ATTRIBUTES
3       key : KeyType;
4       admState : AdministrativeState;
5       opState : OperationalState;
6       userRef : REFERENCE TO UserObject
                  INVERSE OF Bref;
7     PRIMARY KEY key;
8
9     METHODS
10      allocate() RETURNS Boolean;
11
12    PRE-CONDITIONS
13      deleteObject() ONLY IF
        admState = locked;
14
15    PARTY TO ResourceAndUser;
16END;
```

Fig.72

```
17OBJECT TYPE UserObject IS
18    ATTRIBUTES
19        admState : AdministrativeState,
                     PURE VIRTUAL;
20        resourceBstate : OperationalState;
21        Bref : REFERENCE TO ResourceB
                 INVERSE OF UserRef;
22
23    PARTY TO ResourceAndUser;
24END;
```

Fig. 73

```
25DEPENDENCY SCHEMA ResourceAndUser IS
26   FOR ALL UserObject(a), ResourceB(b);
27   RELATIONS a.Bref = b;
28
29   POST-CONDITIONS
30     NOT (a.admState = unlocked AND
31           b.admState = locked)
32     RULES
33      WHEN COMMIT THEN CHECK CONSISTENCY;
34
35   PROPAGATIONS
36     WHEN b.opState = NewValue
37     THEN CONCLUDE a.resourceBstate=NewValue;
38
39END;
```

Fig. 74

```
40OBJECT TYPE ResourceA IS
41   BASE UserObject;
42
43   ATTRIBUTES
44     key : KeyType;
45     admState : AdministrativeState;
46     DERIVED opState = IF (InternalOpState=disabled OR
47                           resourceBstate=disabled)
48                       THEN disabled
49                       ELSE enabled;
50
51     internalOpState : OperationalState PRIVATE;
52
53   PRIMARY KEY key;
54
55   METHODS
56     allocate() RETURNS Boolean;
57
58   PRE-CONDITIONS
59     deleteOblect() ONLY IF admState = locked;
60
61   POST-CONDITIONS
62     NOT (admState = unlocked AND Bref = NULL);
63
64END;
```

Fig.75

```
65 ifndef _UserObject_hh_
66 define _UserObject_hh_
67
68 include <PredefinedTypes.hh>
69 include "MOstateTypes.hh"
70
71class ResourceB
72
73// OBJECT Type Userobject
74
75class Userobject
76(
77public
78   static UserObject* open(Mode,const
79   KeyType&, DbTransaction* transaction=
     NULL);
80   virtual deleteObject() =0;
81   virtual Boolean checkConsistency
     (ErrorMessage&);
82
83   virtual AdministrativeState
     getAdmState() =0;
84   virtual void setAdmState
     (const AdministrativeState)==;
85   OperationalState getResourceBstate();
86   setResourceBstate(const
     OperationalState);
87   ResourceB* getBref();
88   void setBref(ResourceB*);
89
90private:
91.....
92.....
93);
94
95 endif
```

Fig. 76

```
97 include "UserObject.hh"
98 include "ResourceB.hh"
99
100//////////////////////
101//
102// UserObject: METHOD checkConsistency
103//
104
105Boolean UserObject::checkConsistency(
            ErrorMessage& message)
106(
107   Boolean flag = true;
108
109   flag = !(getAdmState()  unlocked &&
110              getBref()->get admState()==locked);
111   if ( !flag) (
112      createErrorMessage(1,message);
113   )
114   return flag;
115)
116
```

Fig.77

```
117 ifndef __ResourceB_hh_
118 define _ResourceB_hh_
119
120 include <PredefinedTypes.hh>
121 include "MOstateTypes.hh"
122
123class UserObject;
124
125// OBJECT TYPE ResourceB
126
127class ResourceB
128(
129public
130   void deletobject();
131
132   void deleteObject();
133   Boolean allocate();
134   virtual Boolean checkConsistency
      (ErrorMessage&);
135
136   KeyType getKey();
137   AdministrativeState getAdmState();
138   void setAdmState (const
      AdministrativeState);
139   OperationalState getOpState();
140   void setOpState(const OperationalState);
141   UserObject* getUserRef();
142   void setUserRef(UserObject*);
143
144private
145   void propagateOpState
      (const OperationalState);
146   void opState (const OperationalState);
147   .....
148   .....
149);
150
151 endif
```

Fig.78

```
153 include "ResourceB.hh"
154 include "UserObject.hh"
155
156//////////////////////
157//
158// ResourceB: METHOD setOpState
159//
160
161void ResourceB: :setOpState(
            const OperationalState newValue)
162(
163  OperationalState oldValue = getOpState();
164
165  opState(newValue);
166  if (newValue != oldValue)
167  (
168     propagateOpState(newValue);
169  )
170)
171
172//////////////////////////
173//
174// ResourceB: METHOD PropagateOpState
175//
176
177void ResourceB: :propagateOpState(
              const OperationalState newValue)
178(
179  getUserRef()->resourceBstate(newValue)
180)
181
```

Fig. 79

```
182 ifndef _ResourceA_hh_
183 define _ResourceA_hh_
184
185 include <PredefinedTypes.hh>
186 include "UserObject.hh"
187
188// OBJECT TYPE ResourceA
189
190class ResourceA : public UserObject
191(
192public
193   static ResourceA* open(Mode,
194   const KeyType&, DbTransaction*
      transaction=NULL);
195   void deleteObject ();
196   virtual Boolean checkConsistency
      (ErrorMessage&);
197
198   Boolean allocate();
199
200   KeyType getKey();
201   AdministrativeState getAdmState();
202   void setAdmState (const
      AdministrativeState);
203   OperationalState getOpState();
204
205private
206   OperationalState getInternalOpState();
207   void setInternalOpState (const
      OperationalState);
208   .....
209   .....
210);
211
212 endif
213
```

Fig.80

```
214 include "ResourceA.hh"
215
216//////////////////////
217//
218// ResourceA: METHOD getOpState
219//
220
221OperationalState ResourceA: :getOpState()
222(
223   if  (get InternalOpState() ==disabled
224         getResourceBstate() ==disabled)
225   (
226    return disabled;
227   )
228   else
229   (
230    return enabled;
231   )
232)
233
```

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SE 93/00687

A. CLASSIFICATION OF SUBJECT MATTER

IPC5: H04L 12/24, H04M 3/24
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC5: H04L, H04M, H04Q

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

SE,DK,FI,NO classes as above

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US, A, 4782506 (MAXIMILIAN SEVCIK), 1 November 1988 (01.11.88), column 1, line 51 - column 3, line 53, figures 1,2	71
A		1-70,72-118
	--	
A	EP, A2, 0442809 (DIGITAL EQUIPMENT CORPORATION), 21 August 1991 (21.08.91), page 2, line 21 - line 29; page 3, line 28 - line 55, figure 1	1-70,72-118
	--	

[X] Further documents are listed in the continuation of Box C. [X] See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" erlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance: the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance: the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
17 December 1993	28 -12- 1993
Name and mailing address of the ISA/ Swedish Patent Office Box 5055, S-102 42 STOCKHOLM Facsimile No. +46 8 666 02 86	Authorized officer Göran Magnusson Telephone No. +46 8 782 25 00